4-1-2020 12:00 PM

Using a systems approach to analyze the operational safety of dams

Leanna M. King
The University of Western Ontario

Supervisor
Simonovic, Slobodan P.
The University of Western Ontario

Graduate Program in Civil and Environmental Engineering
A thesis submitted in partial fulfillment of the requirements for the degree in Doctor of Philosophy

Abstract

Dam systems are arrangements of interacting components that store and convey water for beneficial purposes. Dam failures are associated with extreme consequences to human life, the environment and the economy. Existing techniques for dam safety analysis tend to focus on verifying system performance at the edge of the design envelope. In analyzing the events which occur within the design envelope, linear chain-of-events models are often used to analyze the potential outcomes for the system. These chain-of-events models require that combinations of conditions are identified at the outset of the analysis, which can be very cumbersome given the number of physically possible combinations. Additional complications arising from feedback behaviour and time are not easily overcome using existing tools. Recent work in the industry has begun to focus on systems approaches to the problem, especially stochastic simulation. Given current computational abilities, stochastic simulation may not be capable of analyzing combinations of events that have a low combined probability but potentially extreme consequences. This research focuses on developing and implementing a methodology that dynamically characterizes combinations of component operating states and their potential impacts on dam safety. Automated generation of scenarios is achieved through the use of a component operating states database that defines all possible combinations of component states (scenarios) using combinatorics. A Deterministic Monte Carlo simulation framework systematically characterizes each scenario through a number of iterations that vary adverse operating state timing, impacts and inflows. Component interactions and feedbacks are represented within the system dynamics simulation model. Simulation outcomes provide useful indicators for dam operators including conditional failure rates, times to failure, failure inflow thresholds, and reservoir level exceedance frequencies. Dynamic system response can be assessed directly from the simulation outcomes. The scenario results may be useful to dam owners in emergency decision-making to inform response timelines and to justify the allocation of resources. Results may also help inform the development of improved operating strategies or upgrade alternatives that can reduce the impacts of these extreme events. This work offers a significant improvement in the ability to systematically characterize the potential combinations of events and their consequences.

Keywords

Dam safety, system safety, systems modelling, hydropower, dams, simulation, system dynamics, system dynamics simulation, combinatorics, risk assessment

Summary for Lay Audience

This research presents a novel approach to define and characterize potential combinations of events that can impact the ability to safely manage water flow in dam systems. Dam systems consist of infrastructure whose primary purpose is to store and convey water for beneficial purposes, such as power production, water supply, flood control and recreation. The water barrier is the dam itself, and water passages may include gated or ungated spillway systems that release excess flows, diversions, tunnels or penstocks (pipelines) that convey water to power-generating turbines. Another key part of a dam system is the system operator(s). Operators can be a single person or an organization. In some cases the operation of the dam may be automated. Operators make decisions on how to adjust water flow through the dam based on available information, with the goal to safely and economically manage the reservoir. The failure of a dam can cause a major flood, potentially having catastrophic consequences to human life, the environment and the economy. One possible way in which a dam can fail is through the inadequate control of water flow. For example, should the outflow passages fail to function, inflows into the reservoir can cause the water level to rise to critical levels that may result in failure of the dam. This research focuses on the analysis of flow-control in dam systems. A dam system and the interactions amongst its components are modelled in detail and an exhaustive list of possible combinations of events is developed. Each of these combinations is simulated many times to characterize the potential outcomes that may occur. The simulation model calculates the water levels and flow releases as they change over time. Parameters were developed to provide some indication about the potential impacts of a scenario. The result is a systematic characterization of these unlikely, yet potentially hazardous, combinations of events that can affect the ability to safely operate a dam system. The information produced through this methodology may be useful in developing operating strategies and emergency response plans that could occur over the course of a dam's lifetime.

Dedicated to my hero, my Dad.

Acknowledgments

First and foremost, I would like to thank my supervisor Prof. Slobodan Simonovic. Prof. Simonovic has been a mentor of mine for many years, beginning during my undergraduate degree. He provided guidance for my undergraduate and M.E.Sc theses, and even went so far as to help me in my job search following graduation. He encouraged me to pursue a career in an area that would allow me to develop further as a researcher and for that I am forever grateful. I am very fortunate that Prof. Simonovic would take the time to meet with me weekly throughout the duration of my PhD program, and ensured I received any help I needed with some of the more technical aspects of this work. The technical advice, guidance and direction he has provided has been invaluable to this project. I am so grateful for his unwavering support and encouragement throughout this process.

I would also like to extend my sincere gratitude to our industry collaborator for the project, Dr. Des Hartford. I first met Des during my initial job interview at BC Hydro and he immediately began talking to me about the project that would eventually inspire this PhD research. Throughout my time at BC Hydro, Des has been an excellent and supportive mentor. He was instrumental in securing funding from BC Hydro to complete this research. Des has been the key industry advisor for this work, attending regular meetings over the course of the project to help guide the research direction to ensure it maintains practical relevance. He has contributed an inordinate amount of time evaluating, challenging and discussing this research with me. The invaluable insights gained through his efforts have helped me substantially improve the quality and applicability of this work. Des has also provided guidance which has helped improve the presentation of this thesis as well as address some of the methodology limitations. I look forward to continued collaboration with Des in future research endeavours.

I would like to thank Derek Sakamoto of BC Hydro for his very valuable technical advice and additional guidance on this project. Derek used his own personal time away from work to attend project meetings and was always just an e-mail away from answering any technical questions I had. Thanks also to Dr. Zoran Micovic and Dr. Georg Jost from BC Hydro for their help with some of the hydrological modelling aspects of this work. I would also like to

acknowledge all of my colleagues at BC Hydro who supported this research from the beginning, in particular Stephen Rigbey and Dr. Robert Schubak.

I am very thankful to everybody who helped me with the massive undertaking that was the programming side of this research. I am so grateful to have had the opportunity to work with Patrick Breach from FIDS. Patrick provided a significant amount of programming guidance and help. From the very beginning, Patrick helped me switch to the Python programming language, and at several points during the project has provided his personal assistance for some of the complex programming issues I faced. He developed the VenPy Python package which was used to facilitate initial model development, and later the *sdpy* Python package which allowed me to move the entire system dynamics model to Python. This project would have been a lot harder without his help and friendship. Dr. Andre Schardong has also been extremely helpful throughout this research in terms of programming assistance. He assisted me with developing an optimization model that was initially used in the system dynamics model to represent operations planning. He also helped design the database that would become a critical part of scenario development. My sincerest thank you also to Mr. Levon Manukyan who designed the simulation controller for Sharcnet and helped me tackle such a huge computing task. It would not have been possible without his help.

I am also very grateful for the friendship and encouragement of all of my colleagues in FIDS (both past and current), as well as the friends I made in the broader community of students at Western.

I am thankful for the funding provided for this research by both BC Hydro and NSERC through the collaborative research and development grant. I am also thankful to NSERC and OGS for supporting me financially through scholarships.

I would like to express my sincerest gratitude to the faculty at Western University for their teaching and encouragement throughout my PhD program, as well as my Masters and Undergraduate degrees. I am grateful to Dr. Girma Bitsaumlak for his guidance and help regarding the use of Compute Canada systems. Thanks also to the office and administrative staff who have been so accommodating and helpful during the entire duration of my time at Western University, in particular Stephanie Lawrence, Whitney Barrett and Kirsten Edwards.

Finally, I would like to extend my gratitude to those nearest and dearest to me. Thank you to my parents Denise and Graham King for their love and support, as well as my brother Ian King, and sister Stephanie King. I am so lucky to have such a supportive family. Thank you for putting up with me throughout this process and providing me with help and support in every possible way. I am so grateful to have a close circle of people that I can rely on for help and support at any time.

Table of Contents

List of Tables

List of Figures

List of Appendices

Acronyms

ETA	Event Tree Analysis
FERC	Federal Energy Regulatory Commission
FTA	Fault Tree Analysis
FMEA	Failure Modes and Effects Analysis
IDF	Inflow Design Flood
MDE	Maximum Design Earthquake
PFMA	Potential Failure Modes Analysis
PFM	Potential Failure Mode
PLC	Programmable Logic Controller
PMF	Probable Maximum Flood
PMP	Probable Maximum Precipitation
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
STPA	Systems Theoretic Process Analysis
STAMP	Systems Theoretic Accident Model and Processes
USBR	United States Bureau of Reclamation
USACE	United States Army Corps of Engineers

Chapter 1

1 Introduction

This thesis focuses on the development of a new approach to the assessment of dam safety flow control using a systems approach. Concepts from within risk assessment, general systems theory and control system theory are investigated as potentially promising techniques for the assessment of dams as systems. A new methodology is presented which allows for automated generation and simulation of a more complete range of potential operating conditions for the system using a Deterministic Monte Carlo simulation framework with a system dynamics simulation model. System behaviour is quantified directly from the simulation outputs and helps identify combinations of events which can lead to the failure of dam systems to safely control inflows.

This chapter contains an introduction to dam systems and a historical overview of dam failures. An introduction to the systems approach is also provided here, as well as research objectives and conclusions.

1.1 Dam systems

Dams are highly complex systems containing arrangements of components which interact to store and convey water for one or more purposes, including hydroelectric power, flood control, mine tailings impoundment, and water supply for residential, agricultural or industrial purposes. Dams create reservoirs and use of their storage provides for the redistribution of inflow in time and space. These systems contain physical infrastructure, mechanical components, electrical components, communications equipment and human controllers which are all functioning together for a single purpose: the safe and economical storage and passage of water. The components that influence the behaviour of a dam system can be both physical (eg. infrastructure), or nonphysical (eg. operational decision making). Typical components of a dam system can be grouped into categories of: (1) Infrastructure components such as the dam, penstocks, spillways, gates, turbines, etc., (2) Actuators which are typically mechanical or electrical assemblies that make changes to infrastructure positioning either manually or automatically, remotely or on-site, (3) Operators which

include human or automated system controllers as well as institutional and organizational operating guidelines and rules, and (4) Sensory components such as Supervisory Control and Data Acquisition (SCADA) systems or visual observation. Dam systems have external inputs, such as reservoir inflows and various disturbances, and system outputs or products. Products of a dam system can include reservoir outflow, power generation, environmental or recreational flows, flood control and irrigation water supply. Figure 1-1 contains a labelled photograph of Revelstoke Dam in British Columbia, Canada, which has a number of features that are discussed in the following paragraphs.

The key feature of a dam system is the dam itself, which acts as a barrier to the natural course of a stream or river. Dam structures may be constructed of a variety of different materials in a variety of different ways, and this choice is dependent on the purpose for which the dam will serve as well as the geological conditions in the vicinity of the dam and the availability of construction materials. Materials used in the construction of dams can include timber, concrete, masonry, steel, as well as earth or rockfill in the form of embankment dams (Jansen 1983). Some dam sites may have multiple dam structures, with auxiliary structures known as “saddle dams” that also act to retain the water in the reservoir. For large dams, concrete and earthfill structures (or a combination of these) are most common. There are a number of types of concrete dams, including concrete gravity, arch and buttress dams (Jansen 1983). Earthfill dams may also come in a variety of forms and may be homogeneous (one material makes up the entire dam) or have zones of different fill materials with engineer-specified parameters designed to control seepage and hydraulic gradients. Dams may also have provisions for foundation seepage control such as cutoffs or grout curtains that increase the seepage path to prevent the erosion of foundation materials (Jansen 1983). In Figure 1-1, Revelstoke Dam, British Columbia, Canada consists of a concrete gravity dam and an earthfill dam.

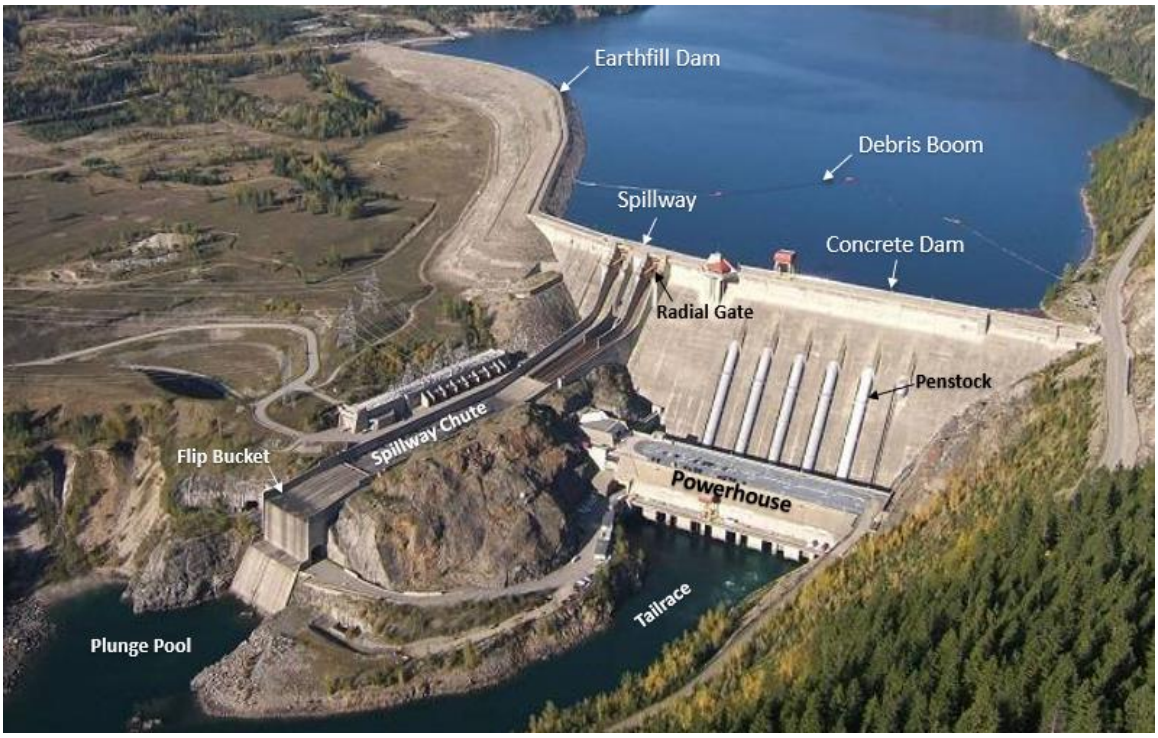


Figure 1-1: Revelstoke Dam, British Columbia, Canada

In addition to the water barrier, dams are typically equipped with some sort of outlet structure to pass the water downstream. In the simplest case of a free overflow weir, water flows over the top of the structure and down the natural course of the river. More complex dams often involve a number of outlets, which can include free overflow spillways, spillway gates, low level outlets and turbines (Jansen 1983). Free overflow spillways are sometimes a lowered section of the dam that is equipped to pass water when the reservoir exceeds the elevation of the spillway crest. The amount of water passing over the free overflow spillway is a direct function of the level of the reservoir. These uncontrolled release structures sometimes involve a chute to direct water downstream. Spillway chutes may be unlined or lined with a material such as concrete (Jansen 1983). Spillway gates and low level outlets are mechanically controlled structures (typically gates or valves) which can be opened and closed to release the desired amount of water. Spillway gates and valves may direct water into a chute, if there is a considerable distance for the water to pass or may discharge water directly downstream of the opening. Each gate or valve has its own rating curve, which is the numerical relationship between gate opening, gate position and

reservoir level (USBR 1987). Gates require many different components to operate, including structural, mechanical and electrical – and in some cases can be operated automatically or remotely as well as onsite. In Figure 1-1, a gated spillway is shown, with two radial gates that discharge into a concrete chute, terminating in a flip bucket and plunge pool.

In the case of hydropower dam systems, another key component of the system is the hydropower generating infrastructure. Intake gates are sometimes used to control the flow of water towards the generating units from the upstream end. Water passes into a power conduit, typically a tunnel or penstock depending on the application, and moves downstream towards the turbine (Komey, 2014). Penstocks are large pipelines, which may be constructed from steel, woodstave, plastic or concrete. Surge shafts are sometimes used to regulate pressure transients in the penstock, which can fluctuate significantly due to adjustments to the turbine flow or closure of valves along the power conduit. Once the water reaches the end of the penstock, there may be a turbine intake valve which controls the flow and may be closed for maintenance. Past the penstock, water enters the turbine. The generator transfers the rotational energy of the turbine into electrical energy which is then converted unto useable voltage in a switchyard connected to the power grid. Turbines may also be equipped with Pressure Relief Valves, which control pressure transients in the penstock during load rejections where the wicket gates must be suddenly closed (Komey, 2014). At Revelstoke Dam (Figure 1-1), water passes through penstocks to the turbines at the powerhouse.

There are many other features of a dam system which function to monitor, protect and control the dam and outlet structures. In some systems, dam operation is implemented primarily from a control center which may be located far away from the site itself. Operations planning based on an inflow forecast typically takes place off site at the control center. Operations may be implemented in real-time, with hourly instructions and minute-by-minute changes to gate and/or turbine flow. The instructions may be sent out as signals from the control center via satellite or communications towers and are interpreted by Remote Terminal Units (RTU’s), which convert the signals into instructions for outlets and/or turbines and may also function to send information back to the control center

(Komey 2014). These RTU's are part of the SCADA system which collects and distributes information and implements controls. Programmable Logic Controllers (PLC's) are another key part of the SCADA system with a variety of functions, including implementation of the instructions transmitted by the RTU's, implementing controls through a human-machine interface, as well as the collection and analysis of sensory data (Komey, 2014). Dams may have extensive monitoring equipment, including gauges to measure the elevation of the reservoir and positions of the gates, piezometers to measure the water level in dams, and weirs to monitor dam seepage (Jansen 1983; Duscha and Jansen 1988). This information can be recorded manually or collected by a PLC and transmitted to the control center using an RTU.

The key input to a dam system is the reservoir inflows. Inflows are a function of the watershed characteristics, local climate and the hydrologic cycle. In the hydrologic cycle, moist air enters the atmosphere through evaporation and transpiration. As it cools, it condenses to form clouds, which release precipitation. Precipitation can fall in the form of rain or snow, depending on temperatures and ground elevations. Precipitation and snowmelt may contribute to reservoir inflows through runoff (water passing over the land surface) or groundwater (water passing through the sub-surface). Precipitation can vary significantly depending on the time of year as a result of seasonal climate influences. In addition to the amount of precipitation, the geological conditions, ground cover, and topography are significant factors affecting reservoir inflows. In regions closer to the poles, snow melt and the associated increase in inflows is often referred to as the "freshet". The freshet is a period of high inflow resulting, in part, from snowmelt due to increasing temperatures. Freshet inflows may also be affected by heavy rainfall which can speed up the rate of snowmelt. The duration and magnitude of the freshet depends significantly on the regions topography, ground cover and climate. In other parts of the world, there may be wet and dry seasons that affect seasonal inflows. Contributions to inflow from groundwater may be significant depending on the regional geology and climate. In addition to natural inflows, there may also be additional inflows to the system from upstream dam outflows or in some cases, water diversion facilities. Forecasting of the inflows is a key part of safe system operation. There is inherent uncertainty in meteorological forecasts, with forecasting errors generally increasing as inflows increase.

Internal or external disturbances represent another input to dam systems. These include earthquakes, debris accumulation, forest fires, extreme wind and rain, ice storms, ice accumulation, vandalism, rodent activity, human error, component aging, etc. These disturbances may also be considered inputs to the system. Proper management of the system under these circumstances is of critical importance in keeping dam systems safe and preventing losses resulting from failures.

For the remainder of this thesis, a *dam system is defined as all components which interact for the purposes of water storage and conveyance*. This includes all civil, mechanical and electrical infrastructure at a dam site, human operations and decision making, personnel and staffing, site access, sensory and communications equipment, information flow, as well as water in storage and conveyance. The dam system input is the inflow as well as any natural disturbances. The dam system output is the outflows and products of the system such as energy. This research focuses on the analysis of dam system flow control – that is, the safe conveyance of water through the system.

1.2 Dam system failures

The three general modes of failure for various types of dam include (1) internal erosion, which involves the migration of material from an embankment (or abutments) and can lead to weakening of the water barrier and eventually dam breach, (2) instability, which can result from uplift pressures or uneven settlement and can lead to failure by toppling or sliding, and (3) overtopping or flow control failures which result from a loss of control of the reservoir elevation and can potentially lead to failure modes (1) and (2) (Regan 2009).

Different types of dams have different risks, for example an embankment dam has risks relating to slope instability, overtopping or internal/foundation erosion whereas a concrete dam has risks relating to foundation erosion, overtopping and instability due to sliding or overturning (USBR 1987). Dams also have risks relating to the conveyance of water: if one or more of the water conveyance components of the system fail or become blocked, there may be an uncontrolled release of flow or the reservoir could rise to an unsafe level that

could trigger failure of the entire dam (Baecher et al. 2013; Komey et al. 2015). High reservoir levels can result in a number of adverse impacts, including increased seepage, increased foundation or dam uplift pressures that could compromise dam stability, or overtopping of dam structures and/or abutments which can progress to erosion, head-cutting and potentially loss of containment of the reservoir. Further, there are obvious risks relating to the collection, transfer, and use of information to make decisions that will ultimately affect the infrastructure and the risk of dam failure or uncontrolled flow release (Komey et al. 2015). In simple terms, the safety of dams relies on the ability to safely contain and convey flows through the dam system.

A better understanding of how dams fail to operate safely and what the contributing factors are can help practitioners identify potential risks in similar structures and system arrangements. Despite the extreme consequences associated with dam safety incidents, post-event information is often limited to the immediate failure mode or proximate cause and the incident consequences. There are very few detailed accounts of the design, operational decisions and other states of the system that may have contributed to dam safety incidents. Some of the more well-known post event assessments of dam safety incidents include Teton Dam (Jansen 1983; Seed and Duncan 1987), Vajont Dam (Jansen 1983; Genevois and Ghirotti 2005), Baldwin Hills Dam (Jansen 1983), St. Francis Dam (Jansen 1983), Carsington Dam (Kennard and Bromhead 2000), Taum Sauk pumped storage facility (FERC 2006), Folsom Dam (Todd 1999) and Oroville Dam (France et al. 2018a).

There are a variety of different resources for information about dam safety incidents, mostly from American organizations. The Association of State Dam Safety Officials (ASDSO) has a number of sources for information about dam failures in the United States, including a website with case studies and lessons learned about select incidents (ASDSO, 2016) and a table containing basic information relating to 187 incidents (ASDSO, n.d.). Jansen (1983) provided an excellent review of dam failure case histories in a technical publication by the United States Bureau of Reclamation (USBR). The National Performance of Dams Program (NPDP) is a database of American dam incidents and failures developed by Stanford University. Most of the incidents are from the late 1900s but the database includes incidents ranging from 1848 to 2015. A wide variety of incidents

are covered, from issues discovered during safety inspections to flow control incidents to complete dam failures, with a total of 2977 incidents. Other researchers have compiled and assessed similar databases to draw conclusions about dam safety risks (Foster et al. 2000a; Zhang et al. 2007; Charles et al. 2011). Regan (2009) compiled a database of over 4000 dam failures worldwide, with half of these incidents coming from the United States. The database was assessed to answer questions mainly about the age of the dam at failure as well as the type of dam and the general failure mode (flood, seepage/piping, structural). Fry et al. (2004) developed a web-based Dam Accident DataBase (DADB) with 900 incidents. Database entries included basic information about the dam and breach characteristics, dates of construction and failure as well as the failure mode, with links to references for users. Analysis of the past failures has shown that internal or foundation erosion and flooding events (overtopping) are the two major causes of catastrophic dam failure (Foster et al. 2000b; Donnelly 2005; Regan 2009).

The likelihood of failure by internal erosion is traditionally estimated using empirical criteria developed by Foster et al. (2000b) and Foster and Fell (2001). Internal erosion processes are generally not well understood, and there are ongoing efforts to better understand the physical processes. Assessment of failure by overtopping is also a complicated process, because of the sheer number of factors which can contribute to the likelihood of dam overtopping. Such factors can include but are not limited to inflows, operational decisions, gate reliability, personnel availability, site accessibility, and natural disturbances such as ice or debris buildup (Lewin et al. 2003; Regan 2010; Komey et al. 2015). There is a large amount of literature detailing various approaches to extreme flood estimation (Bocchiola et al. 2003; Kwon and Moon 2006; Kuo et al. 2007; USBR and USACE 2012), and many risk assessment methodologies check dam performance under these extreme flood loads (Regan 2010; Komey et al. 2015). However, it is difficult to assess overtopping incidents that could happen within the design envelope of the dam system due to a combination of events which together prevent water from being released and allow the reservoir to rise to an unsafe level (Regan 2010).

Hartford et al. (2016) describes the “uncommon combination of common events”, where multiple seemingly benign events combine together to become a significant dam safety

problem. In the Noppikoski Dam failure incident, a number of conditions contributed to an inability to pass flows through the system, leading to overtopping failure of the dam. The mechanical hoist equipment did not function, and a crane was unable to be mobilized to the site in time to remove the stoplogs from the spillway as a result of extreme weather conditions (loss of access) and site staff unavailability. These issues, combined with higher than normal inflows, lead to rising reservoir levels which eventually overtopped and failed the earthfill embankment. Taum Sauk is another example of combinations of events interacting with disastrous consequences. The pumped storage facility was overfilled and breached as a result of improperly calibrated reservoir level sensing equipment and differential settlement of the dam crest.

Lewin et al. (2003) analysed several USACE dams and noted that failure to operate the gate on demand would increase the probability of failure of the dam by between 2 to 250 times. Furthermore, dams may also be in an unsafe state *without* complete structural failure of the dam, as a result of uncontrolled flow releases through a failed conduit. Regan (2010) and Baecher et al. (2013) assess several dam failures and uncontrolled flow releases, noting that dam safety incidents are often a result of complex interactions between system components. Both researchers advocate taking a "dams as systems" approach when assessing dam safety risks in order to avoid omission of potentially significant failure modes.

The existing databases relating to dam safety incidents tend to focus more on the proximate causes of the incidents. The database assessments of Foster et al. (2000a, b) and Zhang et al. (2007) focus mainly on internal erosion in embankment dams, looking at embankment design and construction practices. Other assessments (Donnelly 2005; Regan 2010) look at dams in general but don't tend to further decompose the incidents to look at contributing factors such as operational decisions, lack of maintenance, and component failures.

King et al. (2016a) used information from a variety of sources to assess the causes of dam incidents resulting in uncontrolled releases of water. A database of dam incidents was compiled, and dam incidents were decomposed as much as possible to determine the components involved and the contributing factors. Failures were categorized depending on

the type of incident, with incidents grouped into categories based on the following failure modes: overtopping, penstock failure, embankment failure, uncontrolled flow release and other. Internal erosion events were considered to be a design and surveillance issue and as such were not included in the database, which intended to focus on operational safety. The database also recorded the type of component involved: Mechanical, electrical, structural, operational, and supervisory control and data acquisition (SCADA) systems. Incidents relating to a certain type of component could then be broken down more specifically (eg. for structural the spillway chute, the dam, the penstock, or the gate). Maintenance was considered to be an operational issue and as such is recorded in that category. Other relevant factors relating to the incident were also recorded, for example the presence of disturbances such as debris buildup, landslides, vandalism and earthquakes.

Based on the completeness of information, incidents were then categorized as either acceptable or incomplete. Incomplete sources were omitted in some of the more detailed figures to allow for a more accurate assessment of the proportions of various types of contributing factors. Once all information for an incident was collected, the quality of information was assessed (acceptable or incomplete) and the incident was assigned a rank based on its severity using the following guidelines:

- Catastrophic: Complete loss of flow control
- Major: Temporary disablement of hydraulic structures leading to temporary loss of flow control
- Minor: Temporary disablement of hydraulic structures that could potentially have resulted in a loss of flow control

Table 1-1 (King et al., 2016) contains a list of the different sources used to compile the database. The number of incidents contributed to by each source is also listed. Most of the dams considered are in the United States because this data was most easily accessible. In the future, accessing the DADB of Fry et al. (2004), which was developed by a European team, could help increase the number of dams considered outside of the United States. Figure 1-2 (King et al., 2016) contains a breakdown of the source quality for each source used. The data from NPDP was scaled by 10 in the figure to make the quality of the other sources more clearly visible.

Table 1-1: Sources of incident information for database development (King et al., 2016)

Source	Number of Incidents
NPDP (2016)	1018
ASDSO (2010)	79
USBR (2014a)	17
Charles (2011)	10
Tavakoli (2015)	8
Chanson (2000)	5
FEMA and NJOEM (2004)	5
Van Niekerk and Viljoen (2005)	3
Other	37

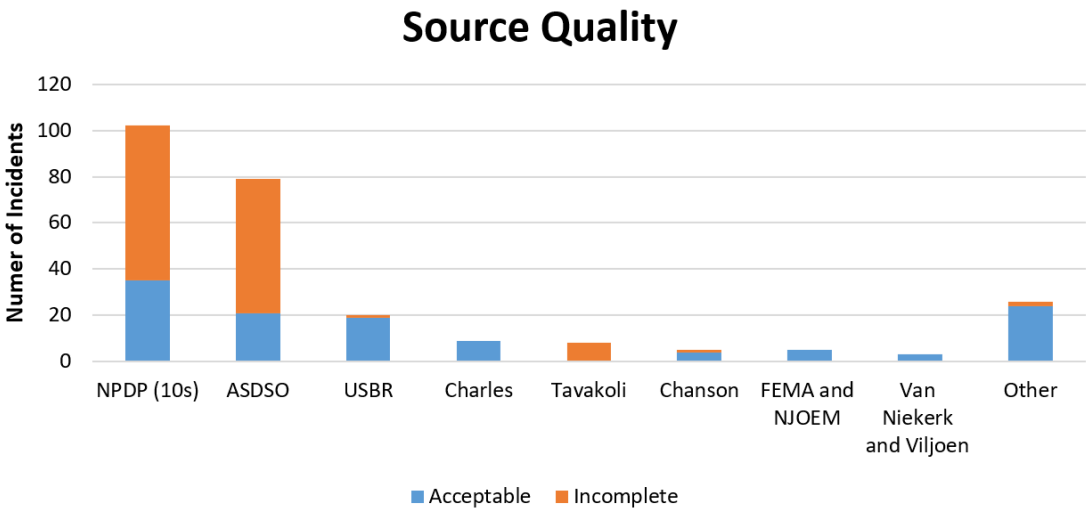


Figure 1-2: Sources used in dam incidents database (King et al., 2016)

A pie-chart of incident types is shown in Figure 1-3 (King et al., 2016), considering all incidents in the database. It is clear from the data that the most common incident type in the database is overtopping. It should be noted that overtopping can be related to many other factors such as operational decision making, gate failures, turbine failures, ice and debris buildup, etc.

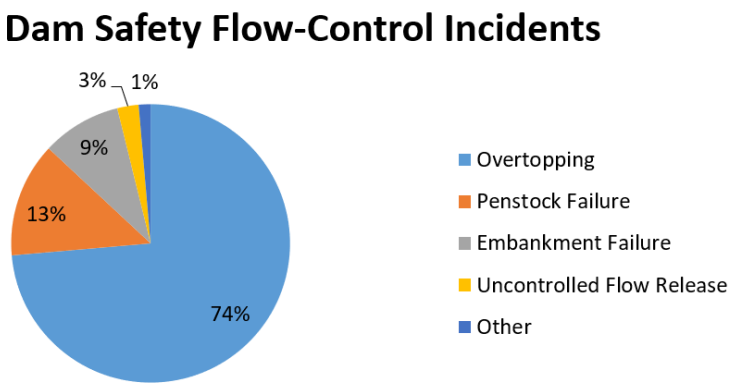


Figure 1-3: Dam safety flow control incidents, by incident category (King et al., 2016)

Figure 1-4 (King et al., 2016) contains a plot showing the factors which contributed to overtopping events, taking into consideration only incidents with an acceptable amount of information. The most common reason for overtopping events is due to lack of spillway capacity. Over half of the overtopping incidents in the database were due to insufficient spillway capacity. However, this could be an indication that the operator did not leave enough freeboard in the reservoir to accommodate inflows up to the probable maximum flood volume. A more detailed analysis of each incident would be required to determine whether this was the case – such information is often not available or not reported. The second most common reason for dam overtopping was a result of a blocked spillway (eg. ice or debris). The next most common contributors to overtopping are gate issues and operator errors. Operator errors could involve the operator deciding not to open the gate, opening it too late or opening it to the wrong position.

Figure 1-5 (King et al., 2016) contains a pie chart of the components involved in the incident, taking into consideration all events in the database that had enough information (677 incidents). Structural incidents were by far the most common, followed by mechanical and operational. Fewer events were related to electrical or SCADA failures. This distribution could be a limitation of the data: because issues relating to electrical and mechanical components may be more quickly resolved and thus less likely to lead to major incidents, it is possible that events relating to these components are under-reported.

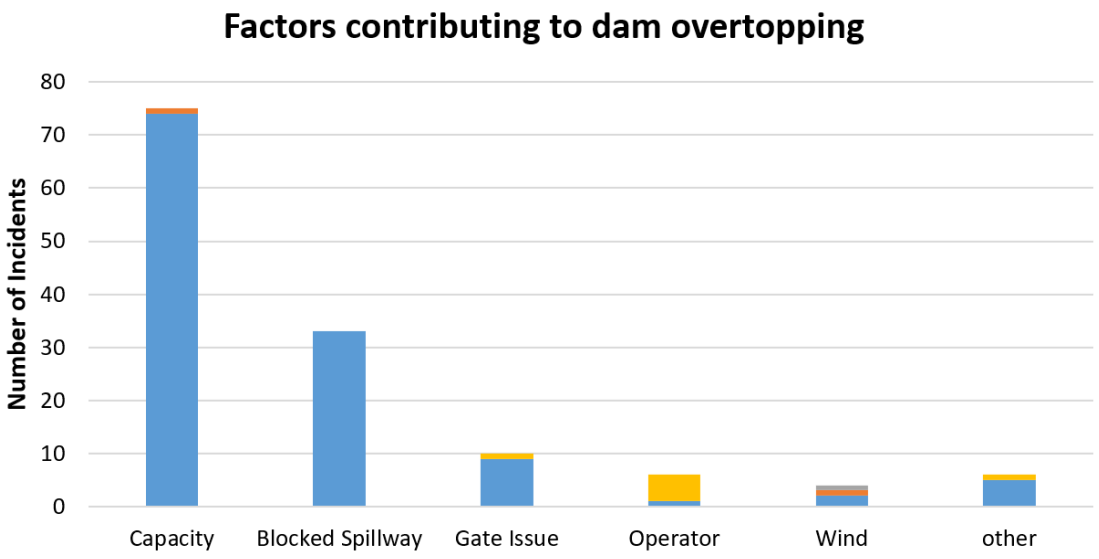


Figure 1-4: Factors contributing to dam overtopping (King et al., 2016)

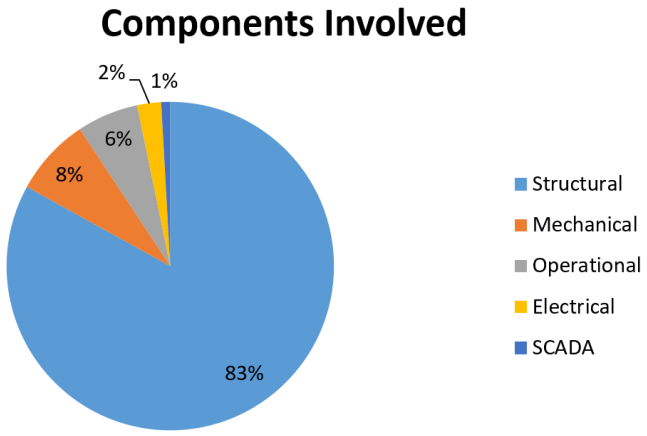


Figure 1-5: Components involved in dam incidents (King et al., 2016)

Figure 1-6 (King et al., 2016) contains a plot of the components involved in structural-related dam safety incidents. Incidents are divided in each category to show the proportion that were catastrophic, major and minor. The most common type of structural incident was an inadequate spillway capacity; most of these incidents result in complete loss of control (the reservoir overtops the dam) and are thus classified as catastrophic. It should again be noted that inadequate spillway capacity may be tied to lack of conservatism in reservoir

operations. The second and third most common structural incidents were related to the spillway chute and penstock, respectively. Because these components actively pass water, they can become deteriorated and may fail if not properly maintained. For the spillway chute failures, not all were classified as catastrophic (uncontrolled flow release) because in some cases the gates could be closed or the reservoir level fell below the sill and the chute could be repaired. Most of the penstock failures are catastrophic because in some systems the intake gates may not be able to be closed under rupture flows. Penstock intake sills are also lower in comparison to spillway sills and therefore significantly more reservoir volume may be released in the event of penstock failure. Structural dam failures and spillway gate failures were the next most common structural flow control incidents. It should be noted that internal erosion and foundation failures were removed from the database and would influence the number of structural dam failures in the figure. Outlet pipes and intake structures were the least common components involved in structural flow control incidents.

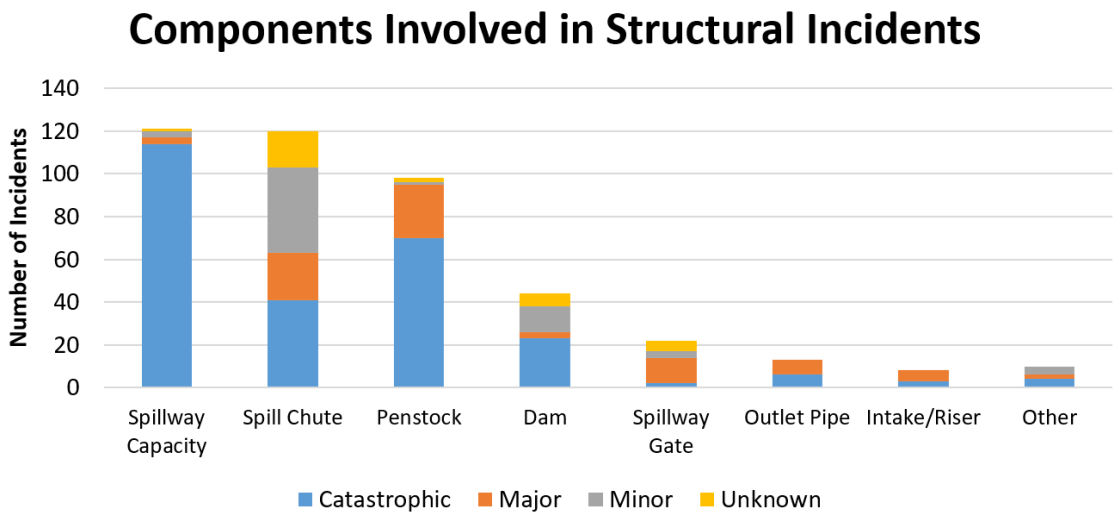


Figure 1-6: Components involved in structural incidents (King et al., 2016)

Figure 1-7 (King et al., 2016) contains a graph showing the mechanical components involved in spillway related incidents. Spillway gate issues were by far the most common, followed by low level outlet and penstock valve failures. Less common were mechanical issues associated with gates, turbines and siphons. It is likely that turbine related issues are

under-reported because forced turbine outages may happen at any power generating facility without impeding the ability of the dam system to operate safely. In some cases, however, turbine outages during high inflow events could lead to potential loss of flow control. It is interesting to note that these incidents did not involve a high number of catastrophic events in comparison with the structural incidents. This is likely due to the fact that mechanical issues can be more rapidly repaired than structural incidents.

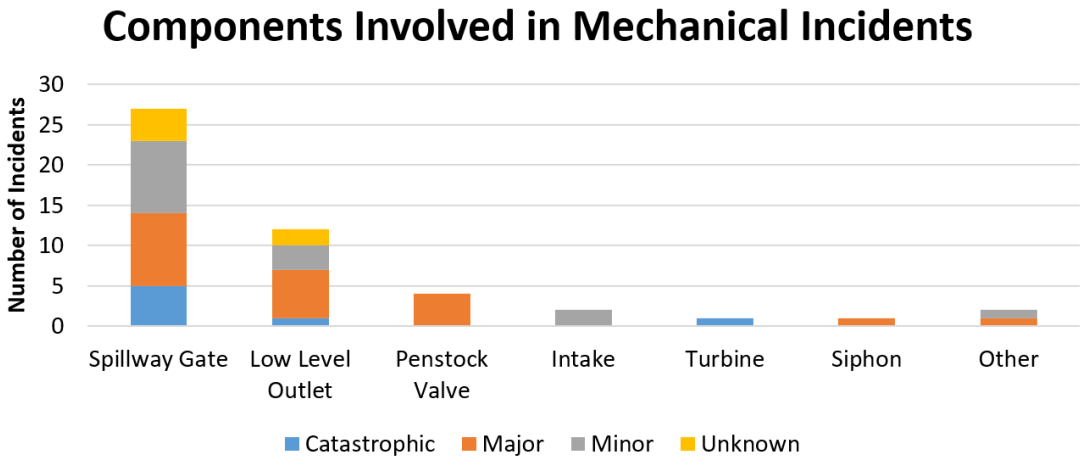


Figure 1-7: Components involved in mechanics incidents (King et al., 2016)

Figure 1-8 (King et al., 2016) contains a plot of the operational factors which contribute to flow control incidents. The data shows that maintenance issues were the most common operational factor, followed by the wrong decision being made. There were less instances of implementation errors or late decisions. It is important to note that operational factors are likely under-reported. Dam operators are not likely to admit mistakes following an event for liability reasons. It is also possible that lack of maintenance was a factor in many of the incidents reported under other component categories but wasn't explicitly mentioned in the event synopsis.

There were only 21 total incidents involving electrical issues and these were mostly related to inability to generate power (forced turbine outage) and power outages. It is possible that electrical issues are under-reported because electrical problems can be solved using back-up diesel, battery or mechanical power sources. Turbine-related power issues are also likely underreported as they are less likely to lead to serious dam safety

issues since the grid can often be brought back online relatively quickly. There were also very few issues relating to SCADA systems and it is likely these are also underreported.

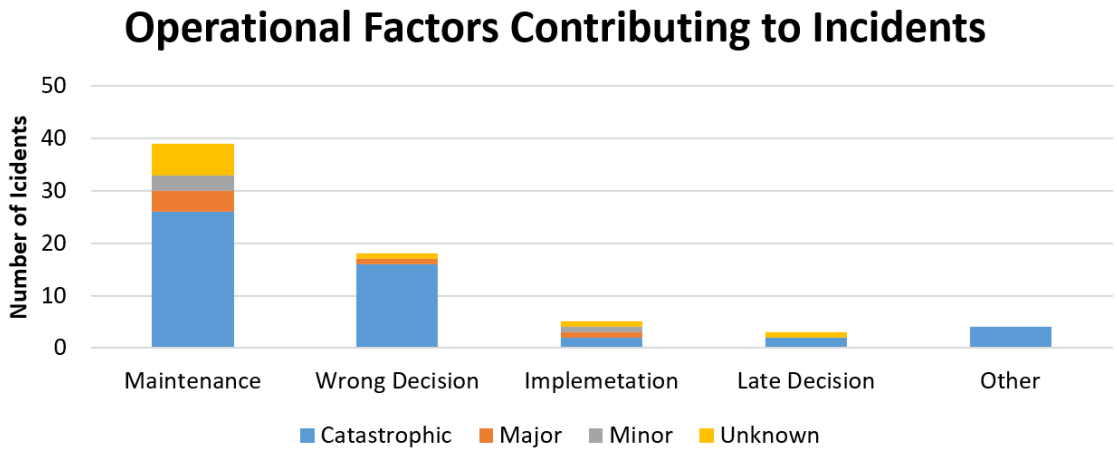


Figure 1-8: Operational factors contributing to incidents (King et al., 2016)

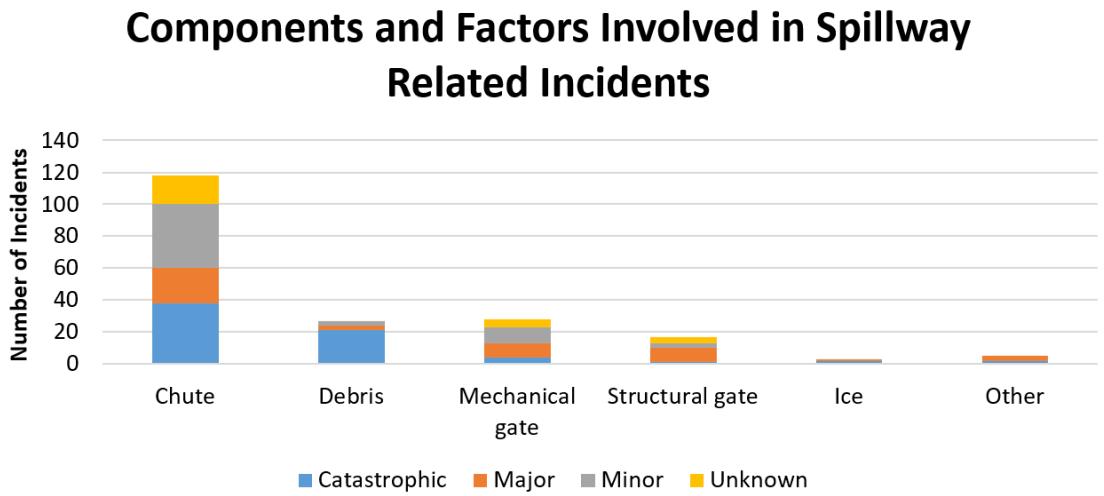


Figure 1-9: Components and factors involved in spillway related incidents (King et al., 2016)

Figure 1-9 (King et al., 2016) shows a plot of the components involved in spillway related incidents. By far the most common component involved in spillway related incidents was the chute. Debris buildup and mechanical gate issues were the second most common issue reported followed by structural gate issues and ice. As discussed earlier, because the sill of the spillway is relatively high, many spillway related incidents are not catastrophic.

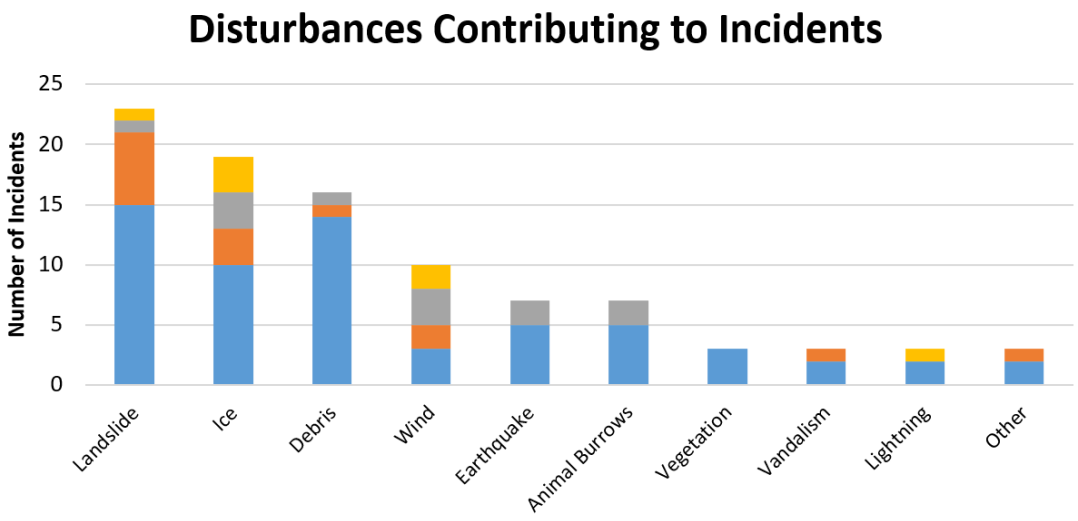


Figure 1-10: Disturbances contributing to incidents (King et al., 2016)

Figure 1-10 (King et al., 2016) shows the various disturbances involved in flow control incidents. The most common type of disturbance was landslides, and many of these were responsible for the penstock rupture incidents. Ice and debris were the next most common, followed by wind, earthquakes and animal burrows. Some of the disturbances are indicative of potential maintenance issues; for example, removal of vegetation, debris and animal burrows should be an important part of any dam safety program. Of all incidents where information about disturbances was available, 7.5% had multiple disturbances contributing to the incident.

The results King et al. (2016a) indicate that there are many factors that influence the ability to control flows in a dam system. As such, dams should be considered (and analyzed) as complex systems of various components working together, quite often known as "system(s) of systems". The sources of information for the King et al. (2016a) study include many non-technical articles which contained limited information about the incidents and as such provide only some insight into the complexity of the incidents and the factors involved. Understanding some of the more detailed event assessments can help illustrate the complexity of the problem of flow control in dam safety. The recent Oroville Dam spillway incident provides some useful context with respect to how a variety of factors may contribute to dam safety incidents. The interplay of components and events within a system can lead to emergent and dynamic behaviour, which the Oroville incident

is a good example of. A brief synopsis and discussion of the incident is described in the following section.

1.2.1 Oroville dam spillway incident

Completed in 1968, the Oroville Dam is a large embankment dam on the Feather River in California and is located at the upstream end of the Oroville-Thermalito complex, which consists of a number of dams and generating stations (FERC 2005a; France et al. 2018a). The Oroville Dam, as shown in Figure 1-11 (France et al. 2018a), consists of an embankment dam, a gated service spillway with eight operating gates and a large concrete chute, an emergency spillway overflow weir discharging into an unlined channel, and the Hyatt Powerplant. There is also a river valve outlet system and a tunnel carrying water towards another generating station downstream.

Figure 1-11: Overview of the Oroville Dam (France et al. 2018)

In February 2017, after severe storms and above average inflows, the gated service spillway was opened and discharged 1400 m^3/s into the chute. On the morning of February 7, 2017, engineers noticed spray coming from the spillway chute and the gate was closed. Upon inspection, a large hole in the foundation and damage to concrete slabs was noticed, as shown in Figure 1-12 (France et al. 2018a). At this point, the reservoir was still rising, and

water began flowing over the emergency spillway overflow weir on February 11, 2017, peaking at around 350 m^3/s on February 12. As the water flowed past the spillway crest structure and onto the natural ground downstream, erosion began to occur, and it progressed through head-cutting, upstream towards the chute structure as shown in Figure 1-13. Undermining of the emergency spillway structure could have resulted in a catastrophic, uncontrolled release of flow. As such, an evacuation order was issued on February 12, 2017 and flow was increased to around 2800 m^3/s over the gated service spillway (France et al. 2018a). This helped lower the reservoir levels to stop flow over the emergency spillway, however it resulted in extensive damage to the service spillway chute as shown in Figure 1-14.

Figure 1-12: Oroville spillway chute damage (France et al. 2018)

Figure 1-13: Erosion downstream of Oroville emergency overflow weir (France et al. 2018)

Figure 1-14: Oroville spillway chute after incident (France et al. 2018)

Following the incident, an independent forensic team (IFT) consisting of experts from various organizations was assembled to review in detail the factors contributing to the incident, providing a detailed report regarding contributing factors and proximate causes of the failure. The immediate cause of the issues in the service spillway chute was uplift pressures that were sufficient to dislodge and remove a section of spillway slab, exposing the underlying foundation directly to high velocity spillway flows. The underlying foundation consisted of rock that was "moderately to highly weathered and even soil like", meaning erosion was able to progress to the degree that additional slab sections both up and downstream of the initial failure were mobilized (France et al. 2018a). For the emergency spillway, erosion began to occur of the natural ground downstream of the spillway structure and in some areas, it began to progress through head-cutting upstream towards the structure. This was mainly a result of the significant depths of erodible

weathered rock and soil as well as hillside topography and insufficient erosion protection and energy dissipation structures.

Management decisions during the events were complicated by a number of issues. Continued erosion of the spillway chute could potentially lead to failure of a transmission tower located beside the spillway. There were uncertainties relating to whether progression of the chute failure upstream could eventually compromise the spillway headgate structure. Debris blockage of the river in combination with spillway tailwater could result in powerplant flooding, presenting potentially long-term issues with water management if the powerhouse was no longer able to discharge flows downstream (France et al. 2018a). Closing the spillway to mitigate these issues would mean utilizing the emergency spillway, the consequences of which were not known at the time. A decision was made to reduce service spillway flows, which resulted in water being released over the emergency spillway and the initiation of erosion there (France et al. 2018a). This presented a new, avoidable, and more threatening issue (undermining of the emergency spillway could progress to dam failure). The system operators were presented with a difficult trade-off and ultimately the decision to reduce flow over the service spillway meant increased flows were necessary later on to prevent further erosion at the emergency spillway.

The IFT report also details extensively many indirect causes of the incident. Several issues in the design and construction of the spillway chute are mentioned, including insufficient foundation preparation for both the chute and emergency spillway, foundation drains which protruded into the chute slab sections, lack of additional reinforcement and robust slab joint keys, and anchor lengths which were insufficient considering the amount of weathered rock on which the chute was constructed (France et al. 2018a). A number of systemic issues relating to organizational, industry and regulatory factors were also identified. Examples of these include a focus on dams instead of spillways, cost control resulting in a reactive approach to managing infrastructure problems, emphasis on dam production ahead of dam safety, as well as overconfidence and complacency regarding the safety of the infrastructure (France et al. 2018a).

The Oroville Dam spillway failure provides insight into the dynamic and often emergent nature of dam safety incidents. There were a large number of direct and indirect factors which contributed to the spillway failures, and management decisions during the incident were complicated by a number of trade-offs. The incident illustrates well the importance of considering the degree of complexity and interactivity in dam systems when analyzing dam safety flow control. This is essential to capture emergent system behaviour, which may not be obvious through analyses of the individual parts.

1.3 Systems Approach

Hartford et al. (2016) advocate for a systems approach to the problem of operational safety in dams and reservoirs, noting that:

> *"A new approach is required, as current engineering practices do not and cannot address the character of some of the most probable causes of incidents and failures, which are the unforeseen combination of rather usual conditions. That is, most incidents and failures occur not because an extreme event occurs (eg. a flood or an earthquake), but rather because a series of more common events occurs, which in their unfortunate and unexpected combination leads to an adverse outcome – an incident or a failure… It may not be possible for an incident or failure to occur if all components, and therefore events, are in a perfectly normal state. Some conditions must be in the range of 'not quite usual' – for example, a 50-year flood, lack of required maintenance, slightly incompetent personnel or organization, bad instructions or policies, a power blackout, or the like – and yet not be extreme or malicious individually"*

Traditional risk assessment approaches include Failure Modes and Effects Analysis (FMEA), Potential Failure Modes Analysis (PFMA), Event Tree Analysis (ETA) and Fault Tree Analysis (FTA). FMEA is a systematic approach to determining the potential failure modes of system components and the effects that these may have on the system as a whole. PFMA is a heuristic failure modes brainstorming technique used commonly within the

dams industry (particularly in the United States). ETA is an inductive, chain-of-events style technique that can be used to determine the potential outcomes from a single initiating event. FTA is a deductive chain-of-events technique that starts with a high-level undesirable event and proceeds in more levels of detail to determine its causes.

Hartford et al. (2016) suggests that systems safety engineering recognizes the three major ways for accidents to occur result from (1) the system capacity being exceeded, (2) combinations of failures of system components, none of which occurring individually would be cause for concern, or (3) a result of complex interactions between system components, none of which may have failed. In traditional dam safety practice, a standards based approach is followed, which addresses the first of these three causes of accidents – checking the system capacity against expected design loads, including extreme floods and earthquakes. Existing risk assessment approaches may provide some insight into the second and third type of accident, however there are a number of shortcomings in this area which are well documented within and outside of the dams industry (Regan 2010; Hartford et al. 2016; King et al. 2016b):

1. The focus of traditional risk analysis tends to be on extreme events at the edge of the design envelope in terms of structural loads and inflows, while failures may occur well within the design envelope due to an uncommon combination of events which individually may not be uncommon (Baecher et al. 2013; Komey 2014; Hartford et al. 2016)

2. Using chain-of-events analyses, all possible system hazards and component operating states must be determined at the beginning of the analysis. This requirement creates immense challenges for the analysis of anything other than simple dam and reservoir systems, since the number of physically possible combinations becomes overwhelmingly large (Hartford et al. 2016)

3. Traditional analysis techniques such as fault trees and event trees often assume a linear progression of events, ignoring component interactions and oversimplifying dynamic system behaviour (Regan 2010; Leveson 2011; Thomas 2012)

4. Events are often assumed to be completely independent of one another, despite the fact that common cause events are possible (Putcha and Patev 2000; Leveson 2011; Komey et al. 2015)

5. Systems are decomposed into more manageable sub-systems for analysis and the interactions between them are completely ignored or simplified (Regan 2010; Leveson 2011; Thomas 2012)

A systems approach is beginning to emerge as a new technique to address some of the aforementioned shortcomings and make progress towards addressing the second and third type of incident. This is discussed in detail within the recent book "Operational Safety of Dams and Reservoirs" by Hartford et al. (2016). The systems approach is fundamentally rooted in systems theory.

Systems theory has a long history of concurrent developments in various fields, with general systems theory being first defined by biologist Ludwig von Bertalanffy (1950; 1968). Von Bertalanffy (1968) defines a system as "complex of interacting elements" in his book dealing with general systems theory and first used the term in a 1950 article dealing with open systems in the fields of physics and biology (Von Bertalanffy 1950). Open systems are those which have inputs and outputs resulting in a change in the system state, whereas closed systems have no external inputs or outputs. The idea of system feedbacks being a function of the system structure is described in this work, with the theory of feedback having origins in cybernetics (Wiener, 1948) and social sciences (Richardson 1991).

Von Bertalanffy (1968) and Forrester (1961, 1969, 1971a) further developed the concepts of systems theory in a series of books dealing with biological, social, economic and other applications. Systems theory, cybernetics and control system theory were at least partly influenced by the efforts of some of the key authors to develop automatic systems for anti-aircraft weaponry and radar devices during World War II (Wiener 1948; Brown and Campbell 1950; Forrester 1989), though Von Bertalanffy (1968) describes several other

key developments which led to a general systems theory. Forrester (1961) began developing system dynamics to analyze industrial and management systems, and pioneered the earliest forms of system dynamics simulation software packages. He later extended the application of system dynamics simulation to model the social dynamics of cities, countries and the world as a whole (Forrester 1969, 1971b, 1989).

Key concepts in general systems theory include the consideration of the system as a whole consisting of interacting parts and the system boundary distinguishing its elements from their surrounding environment. Systems may be as small as a single atom and as large as the universe (Simonovic 2009). Simonovic (2009) provides a more detailed definition of a system as "a collection of various structural and non-structural elements that are collected and organized in such a way as to achieve some specific objective through the control and distribution of material resources, energy and information", and formalizes this as:

$$S: X \rightarrow Y \tag{1}$$

Where X is an input vector, Y is an output vector, and the system is a set of operations that transforms X to Y. Figure 14 (Simonovic 2009) contains a schematic presentation of this definition.

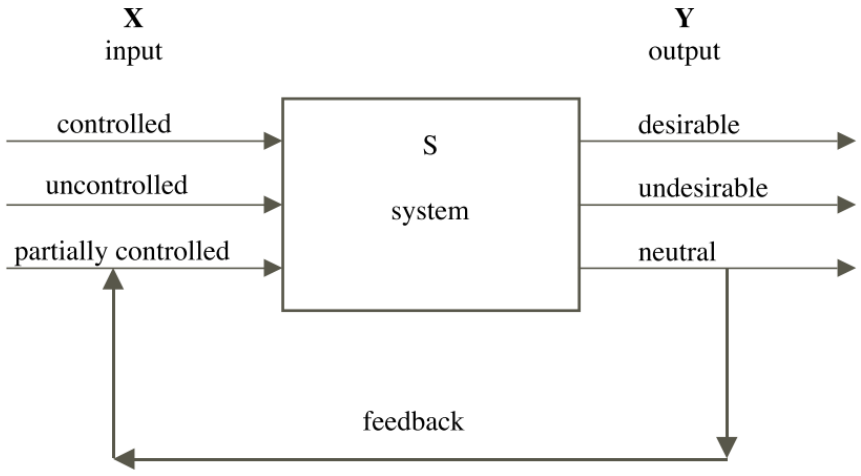


Figure 1-15: Schematic presentation of system definition (Simonovic, 2009)

Another key concept of systems theory is feedback. Open systems, as shown in Figure 1-16 (a) have inputs and outputs that drive the system behaviour. In feedback systems, as shown in Figure 1-16(b), the input is influenced by the system's own past behaviour (Simonovic 2009). The system is able to respond to its outputs by adjusting the inputs. This self-regulating behaviour, known as homeostasis, is present in many mechanical systems and inspired the work of Wiener (1948) on man-machine systems. Wiener (1948) pioneered cybernetics – which is the study of control mechanisms in man-machine systems – and his work introduced the theory of feedback mechanisms, describing a variety of stabilizing and oscillatory systems. In parallel with the concept of feedback being introduced within the field of cybernetics, it was also being described within the context of social systems (Kast and Rosenzweig 1972; Richardson 1991). A thermostat provides an excellent example of self-regulation, where the thermostat is able to adjust the heat production based on the temperature in the room and the desired temperature (Simonovic 2009). Homeostasis acts to steer the system towards some desired goal.

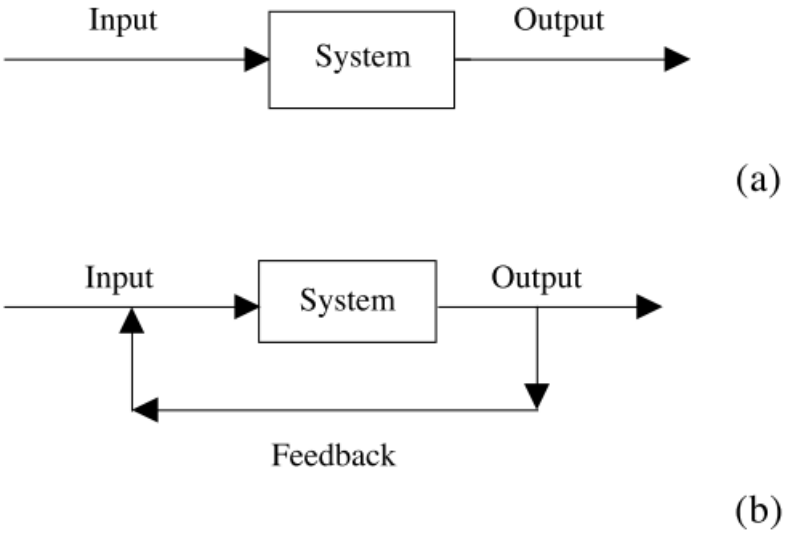


Figure 1-16: Schematic of an open system (a) and a closed system (b)

A feedback loop is a closed path connecting two or more elements of a system. Understanding feedback loops requires an understanding of causality, that is, what elements of the system affect other elements of the system. The two types of loops are (1)

negative or balancing loops, which act to keep the system in a steady state, and (2) positive or reinforcing loops, which reinforce changes to the system with more change. Systems may contain one or potentially many of these loops and can be represented using causal loop diagrams that show the relationships between elements of the system (Forrester 1971a).

Dams may be considered a type of open system, where the inputs consist of system inflows and disturbances, and the outputs consist of system outflows and products (for example electricity). This is shown in Figure 1-17.

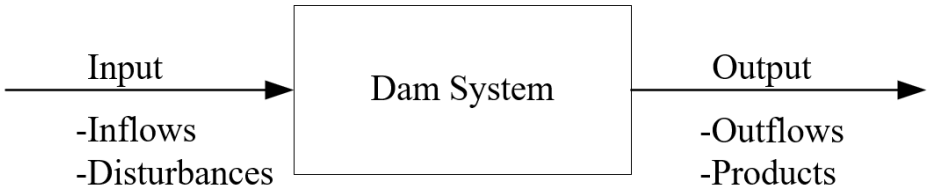


Figure 1-17: Dams as open systems

This configuration indicates limited "self-awareness" that is seen in closed systems. Inflows and disturbances cannot be controlled. However, within the system itself there may be many examples of closed-system type feedback loops present. To model the internal dynamics of the system itself there are a few useful aspects of general systems theory to consider. Control systems theory, which falls under the umbrella of general systems theory (Von Bertalanffy 1968), offers a new way of considering the structure of hydropower systems, which are effectively flow control systems.

Control systems theory arose as a means of designing man-machine feedback systems that self-adjust to achieve the desired outputs (Wiener 1948). Åström and Murray (2008) define control as the use of algorithms and feedback in engineered systems. According to Åström and Murray (2008), controllers act to dynamically adjust the behaviour of system elements to achieve desired system outputs, using feedback to make adjustments. One of the earliest examples of feedback in engineered systems was the development of a centrifugal governor, which controls the throttle of a steam engine to maintain a constant speed (Åström and Murray 2008). The central concept of control systems theory is the use of

feedback loops for sensing, computation and actuation. Leveson (2011) presents a generic feedback control loop which is simplified slightly in Figure 1-18.

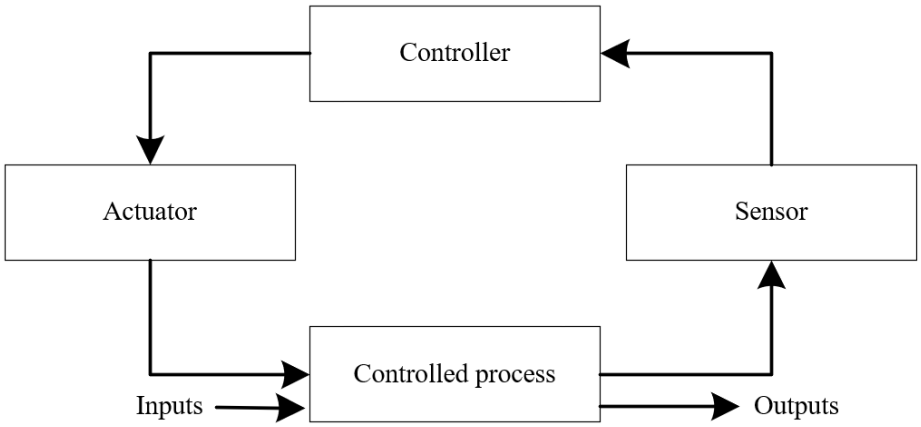


Figure 1-18: Generic control system structure

Considering how the generic control system structure shown in Figure 1-18 can be applied to dam systems is relatively straightforward. The controlled process represents the hydraulic system state, that is, the water barriers, passages and infrastructure on the ground – the dam(s), gate(s) and turbine(s). The system state is measured by sensors – sensors may measure the current reservoir elevation, positions of gates, and even rainfall to predict system inputs (inflows). The controller represents the processing of that information into decisions regarding the required control actions to maintain safety and push the outputs towards the desired level (outputs can be power production as well as outflow). In a dam system, the controller may be a software program, a single person interpreting the system state and making decisions, or multiple people within an organization working together and using mathematical process models to assist in decision making. The output of the controlled process in the feedback loop is a set of control actions, or instructions, that are implemented through actuation of system features that change the hydraulic system state. Actuators in this case are the mechanical-electrical arrangements of infrastructure that function to change the positions of outlet structures (gates and turbine) to modify the outputs and keep the system safe. A control action could be to open the gate (actuate) to a certain position, with the goal of maintaining a safe reservoir level and avoiding excessive flooding downstream of the dam.

Leveson (2011) has applied the concept of control systems to safety in the aerospace industry, developing the Systems Theoretic Accident Modelling Process (STAMP) for accident analysis as well as Systems Theoretic Process Analysis (STPA) for the design and analysis of engineering control systems. These methods provide a systematic process for determining the potential control flaws that can lead to hazards in engineered processes, and they are based on the analysis of the hierarchical control system structure (Leveson 2011). An STPA analysis is essentially a guided brainstorming session whose participants work through the control loop to determine potential control flaws, and further analyze what conditions could possibly lead to them. Identifying the control flaws allows engineers to determine methods to mitigate or eliminate them in system design and improvement.

One fundamental difference limits the potential of the STPA approach when applied to dams. Leveson (2011) presents the approach to analyze control systems operating within the natural environment. Dam systems are systems that both operate within nature *and attempt to control it*. The key issue that complicates the problem of dam safety analysis is that the main system input (inflow) is a completely uncontrollable, nonlinear variable that the system intends to control. Controls may also be active (gates) or passive (free overflow spillways). As such, determining how the system will respond to changes in inflow as well as disturbances (both internal and external) requires a slightly different approach. Characterizing the reservoir elevation in response to the system operating state and inflows is a critical problem for dam safety analysis. While STPA can provide very useful insights regarding the system's structure and its potential vulnerabilities, dynamic analysis of the system response is required to fully understand and mathematically characterize system behaviour.

Two of the major techniques that can be used in the dynamic analysis of systems include simulation, optimization and multiobjective analysis (which expands on optimization to problems with multiple objectives). Simulation involves a "what if" assessment of the various inputs to a system, where outputs are determined in response to a particular set of inputs. Simulation inputs may be varied to determine system behaviour under a range of conditions (Simonovic, 2009) and link the system structure to its behaviour. Optimization, in contrast, provides a single optimal solution to a given system configuration, with

performance measured based on some objective function (eg. maximize profit). Optimization techniques are useful mainly for determining optimal operations strategies driven by a single articulated goal. They are unable to deal with the dynamic, feedback-driven behaviour of complex systems. Optimization techniques may, however, present a useful tool for capturing operator's thought processes and priorities in development of operating instructions. Optimization can be extremely useful for developing optimal operating decisions and policies. Simulation is the most promising systems analysis technique for this research because it (a) facilitates a very detailed representation of system structure, interactions and feedbacks, (b) links the system structure to system behaviour, and (c) allows for the assessment of the dynamic system response to various operating conditions.

System dynamics simulation (Forrester 1971a) is a particularly promising simulation environment to deal with highly complex hydropower dam systems. In system dynamics simulation, the pattern of interaction between system elements is called the system structure, and the behaviour of the system is linked to its underlying structure (the relationships between system components). System behaviour is defined by the way in which the system variables change over time. The dynamics of how a system changes over time can be investigated by changing either the inputs or the system structure (Simonovic, 2009).

In order to carry out a system dynamics simulation, development of a model includes the following steps (Simonovic 2009):

1. Understanding the system and defining its boundaries
2. Identifying the variables that will influence the system's behaviour
3. Using mathematical relationships to describe the relationships between the variables
4. Defining the structure of the model
5. Simulating the model to understand the system behaviour

The building blocks of system dynamics simulation models include (1) state variables (stocks), (2) flows, (3) auxiliary variables and (4) arrows showing relationships between variables which may include delays. The links between these model elements are interactions and feedback loops which ultimately drive the system's behaviour. Stocks are

shown as boxes and represent the state-variables, or variables which increase or decrease in value over time and whose value can only be changed by flows. Flows are represented as rates over time which change the value of a stock. Each auxiliary variable in the model represents either an equation that is a function of the inputs (represented by arrows) or a constant. Delays may be added which represent time lags to variable changes.

The key advantage of system dynamics simulation is the ability for it to be used as a problem-solving method. When problematic patterns of behaviour are observed, the relationships in the model that contribute to the issue can be inspected and the system structure can be modified to potentially eliminate or reduce the problem (Simonovic 2009). The system behaviour contains dynamic information about the state of each model variable, which could be useful for characterizing the reservoir elevation in response to a variety of system loads. System dynamics modelling offers a potential approach for assessment of a wide range of operating scenarios for a dam system, using automatically generated scenarios of potential component operating states. System dynamics is particularly suited for the modelling of control systems. The generic control system of Leveson (2011) is modified to represent a dam system in Figure 1-19, and extended to show the components within the different sub-systems in Figure 1-20. These provide a basic representation of a dam system as defined in this thesis.

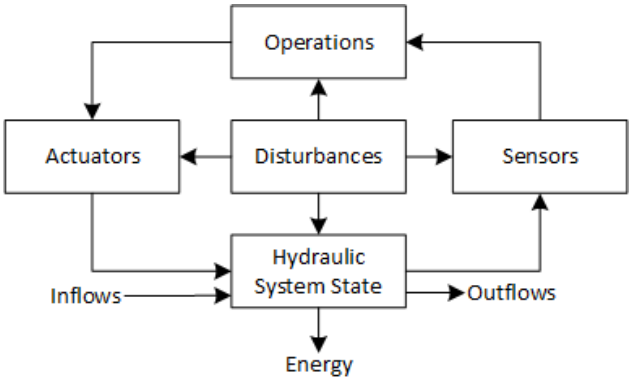


Figure 1-19: Generic control system structure adapted for a hydropower system

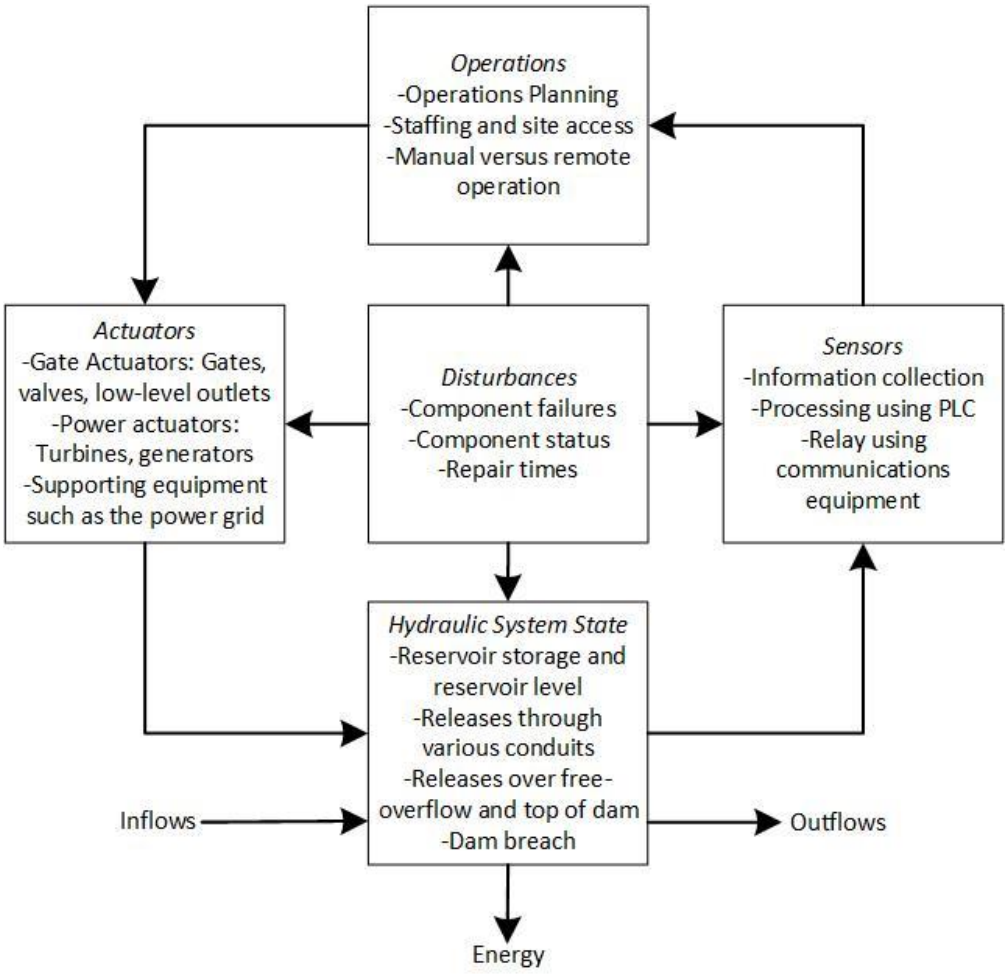


Figure 1-20: Detailed control system structure adapted for a hydropower system

The following section describes the proposed objectives of this research in further detail.

1.4 Research Objectives

The primary objective of this research is to apply systems analysis techniques to the problem of flow control in dam safety. In particular, developing and implementing a methodology that facilitates the characterization of reservoir elevations for particular sets of inflows and operational constraints (scenarios) is necessary. This research draws on aspects of the broad domain of general systems theory as well as risk assessment, with the

goal of providing a systematic and thorough assessment of dam system performance under a wide range of loadings. The research objectives are as follows:

1. Investigate the use of systems analysis and risk assessment concepts from within and outside of the dams industry in terms of their ability to determine potential operating scenarios for dam systems and the impacts scenarios have on system outcomes.

2. Develop an approach that helps define a more complete range of potential operating scenarios (operating constraints) than is possible using existing techniques alone.

3. Develop an improved dam safety analysis methodology that facilitates investigation of all potential operating scenarios and allows for information pertaining to individual scenarios to be analyzed.

4. Develop a simulation approach that can handle complexity in system structure, feedbacks, interactivity and nonlinear behaviour and uses object-oriented modelling to improve model accessibility.

5. Investigate dynamic indicators of system performance with respect to safety, as well as scenario criticality parameters that can be used to rank the importance of various scenarios from the simulation outcomes.

These objectives lead to the development of a methodology that is applied to the Cheakamus Hydropower Project, which is located North of Squamish, British Columbia, Canada (See Figure 4-1). Cheakamus Hydropower Project is a single-reservoir system which discharges water from Daisy Lake through the main dam into the Cheakamus River through two spillway gates and a low level sluice gate. The Cheakamus River is part of the Squamish River catchment and flows into the Squamish River further downstream, eventually discharging into the ocean. Water from the reservoir is also discharged through two hydroelectric generating units in a remotely located powerhouse on the Squamish River, upstream of the Squamish-Cheakamus confluence. This system is modelled in detail and operating scenarios for the system are generated. Due to computational time limitations, a simpler version of the same system is also created, and the scenarios for the

simple system are run through a simulation model to produce a wide range of potential outcomes for the system. A comparison is provided between two different free overflow spillway configurations and operating schemes, to demonstrate how the modelling approach can give insight to dam owners and decision makers in terms of how system modifications affect system safety.

1.5 Research Contributions

The key outcomes of this research, in terms of advancements to the current state-of-knowledge are as follows:

1. It is generally accepted that a complete probabilistic risk assessment of complex dam systems is outside the current state-of-knowledge. This is partly due to the difficulties in estimation of failure probabilities (of the individual components and the systems themselves). This research shifts the focus to assessment of as many possibilities as can be identified, with the goal of providing a complete and indiscriminate assessment of as many possible outcomes for the system as can be generated (improved coverage of the "possibility space"). Probabilistic assessment is possible using the approach presented but is not the focus of the current research. However, the direct outputs from simulation can be used to estimate conditional probabilities of dam overtopping failure and reservoir level exceedance for a particular scenario.

2. Current practices tend to focus on extreme, low probability events such as the Probable Maximum Flood (PMF) and Maximum Design Earthquake (MDE) at the edge of the design envelope, when events well within the design envelope may be contributing more to the overall risk. Assessment of events within the design envelope typically rely on expert judgement for scenario selection with only a small portion of possible scenarios being assessed in detail. There is currently no automated procedure available to determine a complete set of operating scenarios for dam systems. This research proposes a methodology that uses combinatorics to generate a more complete set of potential system operating conditions, including events within and at the edge of the design envelope. The approach presented in

this research automates the procedure of scenario generation, producing an exhaustive list of scenarios which results in slightly reduced subjectivity, though some subjectivity and expert judgement is inherently required in model development and operating state definition.

3. Chain-of-event techniques such as fault trees are commonly used in dam safety assessments. This type of analysis is linear and oversimplified because it is incapable of properly handling component interaction and system feedbacks. The simulation model presented in this research is capable of modelling feedbacks and component interaction, providing a more realistic representation of complex dam systems. Results show how the reservoir level changes with time, which is a key goal of dam safety assessments that is not easily achieved using chain-of-event modelling.

4. The simulation framework presented in this research is capable of a more thorough analysis of all potential scenarios determined through the automated scenario generation. In the Deterministic Monte Carlo Simulation framework, scenarios are the deterministic model inputs. The scenario impacts, timing and inflows can be varied using Monte Carlo techniques to more thoroughly explore the system's "possibility space". This results in estimates of conditional probabilities of failure and reservoir level exceedances over key levels, as well as failure inflow thresholds, which are natural outcomes of the approach presented in this work.

5. The simulation modelling approach presented in this research is easily adaptable and can be modified to experiment with various sets of potential operating rules, response strategies and upgrades. When compared, asset owners and decision makers can quantify how the potential scenario outcomes change as different measures are introduced.

Much of the recent focus on the operational safety of dams and reservoirs involves the utilization of fully stochastic simulation techniques, where probabilities of operating states are defined as inputs and operating states are randomly changed throughout a single continuous simulation. Stochastic simulation is quite useful and efficient for determining

the overall likelihood of flow control failure for a dam system. However, the simulation effort focuses on more likely events, so an extremely large computational effort is required to thoroughly analyze combinations of events. The coverage of the complete "possibility space" is driven by the probabilities of the events being considered and the number of years for which the model is run.

This research proposes a Deterministic Monte Carlo simulation framework to systematically analyze combinations of component operating states. A systematically defined set of possible combinations of operating states (scenarios) is used upfront as a deterministic simulation input. Monte-Carlo variation of operating state outcomes (such as outage lengths, error magnitudes, timing of impacts and inflows) explores each scenario more completely. The key sources of novelty in this work are (a) the automated, combinatorial definition of operating scenarios and (b) the exhaustive exploration of scenario outcomes using a Deterministic Monte Carlo simulation framework. System dynamics simulation modelling is used to execute the simulations. The modelling approach is object oriented, providing a convenient tool for representing complex systems, and is easily modifiable which makes it particularly amenable to optioneering-style assessments. The analysis in this research is performed for each scenario, regardless of scenario likelihood. The influence of initial events on subsequent events is analyzed to ensure scenario outcomes are representative of the input scenario. Useful information can be extracted from each scenario's simulation results. The goal is a more thorough assessment of potential operating scenarios than is possible using traditional risk assessment approaches or stochastic simulation techniques. The Deterministic Monte Carlo approach ensures a more complete coverage of the "possibility space" for a dam system. Complete probabilistic assessment is possible using this approach if information is available to support it. Sensitivity analysis to operating state probabilities is possible without significant additional computational effort (this is a particularly promising direction for future research but is not focused on in the current research).

1.6 Outline of the Thesis

Chapter 2 provides a literature review detailing existing techniques most commonly used for traditional dam safety as well as the relatively new field of dam safety risk assessment. Next, a discussion of the research relating to the shortcomings of the traditional approaches to general risk assessment techniques is provided, including a review of some more recent work meant to reduce these shortcomings. Finally, a discussion of systems analysis techniques is provided, and some conclusions about the capabilities of these techniques are provided.

Chapter 3 contains the complete and detailed methodology used in this work. An overall methodology justification and requirements are presented first. Next, the scenario development is described in two sections relating to the development of the component operating states database and the mathematical formulation to automatically convert database information into operating scenarios. Next, the Deterministic Monte Carlo simulation framework is presented. A description of the system dynamics simulation modelling approach is described, followed by the Monte Carlo techniques used to create scenario iterations. The general simulation framework and steps are presented next as well as a discussion of computational considerations. The following section describes the necessary simulation model input data. Finally, scenario outcome assessment is described.

Chapter 4 contains a description of the case study. First, a description of BC Hydro's Cheakamus Power Project is described, followed by a presentation of the database and scenario generation. Next, a description of scenario generation for a simplified representation of Cheakamus is described, followed by a description of the simplified system dynamics model development, testing and model runs. Simulation model input data is described in the following section. Finally, results are presented.

Chapter 5 contains a discussion of the results and an overall methodology assessment. A discussion of future directions for this research is also provided. References and appendices follow.

Chapter 2

2 Literature Review

A review of the literature relating to traditional dam safety practice and current risk analysis techniques is provided in this chapter. The following section describes traditional dam safety practice. Next, a detailed discussion of risk analysis techniques is provided in the general context as well as within the domain of dam safety. The final section of this chapter contains a discussion of the systems approach to safety.

2.1 Traditional dam safety practice

Traditional dam safety practice typically follows a standards-based approach, where a dam is expected to be capable of passing a certain set of extreme loading conditions, such as the PMF (Mcgrath 2000). There is a significant amount of effort spent on estimating these extreme loading conditions, which are the "edge" of the design envelope. Factors of safety used in the design of the system are checked as more information becomes available and the estimates of these extreme loading conditions are refined. For example, as new flood estimation methodologies and improved hydrometeorological and hydrological data become available, dam owners can use this information to re-calculate the probable maximum precipitation (PMP), which is then used to compute the PMF (USBR 1987; USACE 2011). Similarly, structural, seismic and other load conditions relating to natural disturbances can be refined over time, and the standards-based approach essentially checks and re-checks whether the dam can withstand them.

The United States Bureau of Reclamation USBR has a series of publicly-available standards, a number of which relate to dam and spillway design. The Spillway and Outlet Design Standard (No. 14) deals primarily with flow control in dams. The USBR states that the hydraulic loadings to be considered in spillway and outlet design are the Inflow Design Flood (IDF) and construction diversion floods (USBR 2014b). The IDF is defined as the maximum flood hydrograph used in the design of a dam, and it is either equal to or smaller than the Probable Maximum Flood (PMF). The USBR defines the PMF as "the flood hydrograph that results from the maximum runoff condition due to the most severe

combination of hydrologic and meteorological conditions that are considered reasonably possible for the drainage basin under study" (USBR 2014b). Selection of the IDF in dams built before the early 1940's was based on extrapolation of existing data, for example selecting a flood that was 50% larger than the flood of record for a site (USBR 2014b). Between the late 1940's until the 1980's, IDF's were set equal to the maximum probable floods (roughly equal to the PMF but computed with only site specific data) until the PMF was adopted. IDF selection was then modified to consider the downstream hazard potential classification and possible impacts relating to loss of operation. Eventually, frequency flood hydrograph calculations were used in selection of the IDF, where dams of a particular consequence category were designed to withstand a flood with a particular return period. Since the mid 1990's, a quantitative risk-based approach has been adopted by most American agencies. The USBR utilizes an f-N chart, which is a graph that plots estimated loss of life against the probability of different failure modes. The USBR's f-N chart has defined zones, where points within certain zones of the chart require either increased or decreased justification for further risk reduction. A starting frequency flood is defined and f-N pairs corresponding to it are plotted. The frequency flood is then increased until all of the f-N pairs (for each failure mode) are within the zone indicating decreasing justification to reduce risks. The USBR's design standard also contains a description of potential failure modes (PFM's).

A similar process for IDF calculation is used by the United States Army Corps of Engineers (USACE) and other American dam organizations, and the process is generalized in the Federal Emergency Management Agency's (FEMA) "Federal Guidelines for Dam Safety" (FEMA 2004). The Canadian Dam Association's (CDA) guidelines describe both risk-based and consequence-based approaches for selection of IDF and MDE (CDA 2007). The USACE also provides engineering manuals that include technical guidance and standards. In the "Safety of Dams – Policy and Procedures" document (USACE 2011), minimum requirements for hydrology and hydraulics of dam systems are outlined, including the capacity requirements for spillways and outlet works, as well as the reliability of gates. Geotechnical and structural minimum requirements are also described, and the document cites reference material containing quantitative standards, where applicable. Factors of

safety for structural assessment and design may be provided by the dam agencies, as well as loading conditions to be considered (CDA 2007)

In addition to standards, there is a considerable focus on best practices in dam safety. The USBR and USACE developed a Best Practices Training Manual (USBR and USACE 2015a) containing chapters covering a variety of considerations in dam safety. Of particular relevance to the flow control focus of this work are the chapters relating to overtopping failure, failure of spillway gates and operational considerations. In the chapter on overtopping failure, the manual mentions that vulnerability of the system to gate failures during major floods can be simulated using simple flood routing by eliminating one of the gates from the analysis. The manual also suggests testing the sensitivity of flood routing by implementing delayed control of the gates as a result of human error. The chapter on failure of spillway gates focuses mainly on structural strength and stability assessment, as well as failure modes and best practices for maintenance. The operational risks chapter focuses on issues relating to events within the design envelope of the dam and suggests the use of event trees for their assessment (see Section 2.2.4). There is also a chapter on the probability of failure of electrical and mechanical systems of spillway gates, which suggests a multi-step approach to incorporating this equipment in dam safety analysis. First, probability distributions are applied for estimation of individual component failure rates. Next, fault trees (see Section 2.2.3) are presented as a way of analyzing the probability of failure for the gate system as a whole. Finally, event trees detailing the chain-of-events for the overall dam system are suggested as a means of determining the overall likelihood of failure as a result of failure of the gate to operate on demand. Frequent inspection is suggested as a best practice to identify and address failures of gate equipment. Gate power supply redundancy is also recommended.

In general, the North American dam associations seem to be shifting towards a risk-based approach to dam safety, the key tools of which are described in the following section. In terms of international dam safety practice, McGrath (2000) provides an overview of the use of risk assessment in dam safety, using specific examples of legislature and current practice from several countries. Bowles (1998a) provides a review of the state of the practice based on experience in risk assessment for dams in the U.S. and Australia, noting

several drivers that have lead dam owners to adopt a risk-based approach. One of the key issues is that severe standards may require cost-prohibitive measures for compliance. Bowles (1998a) notes that increasing severity of standards does not always result in reduced risk because dam owners, regulators, and government agencies simply cannot afford to meet the standards. In some states, regulators have worked with dam owners in a risk-based approach which prioritizes projects and partial fixes that are affordable to the dam owner, resulting in an overall reduced risk (Bowles et al. 1998a). Using a risk-based approach, dam owners can provide numerical risk assessment outcomes as justification for focusing on the most significant dam safety risks in the portfolio of dams (Bowles et al. 1998a, b; Bowles 2001). Portfolio risk assessment is a technical ranking method used to prioritize dam upgrades, and has been applied to dams in Australia (Bowles et al. 1998b; Foster et al. 2000b), the U.S. (Cyganiewicz and Smart 2000; USACE 2011; USBR 2011; Srivastava 2013), the U.K. (Morris et al. 2012), Europe and Canada (Donnelly 2005).

It is important to note that the standards-based approach and the risk-based approach are not mutually exclusive, as standards are typically included in the risk-based approach (USACE 2011). Further, the focus in these technical guidelines still seems to be placed on definition of the design envelope and assessment of dam performance at the edge of this design envelope. In essence, the standards-based approach is still in place – the system's design envelope is simply determined using more sophisticated probabilistic risk assessment tools. While some discussion is provided with respect to operational safety under normal conditions (eg. USBR 2014b), there is very little guidance relating to how the operational safety of the systems can be assessed, though event trees are presented as a potential tool. It is worth noting that Canadian structural code associations are contemplating a switch to "performance-based engineering" which uses simulation to assess structural performance risk in response to a range of potential loading conditions (Ellingwood 2017). This shift has resulted from a need to consider climate change impacts in structural design.

The following section details some of the current practices in risk analysis, from within and outside of the dams industry.

2.2 Current practices in risk analysis

Risk is most frequently defined as the product of the failure probabilities and consequences. Risk analysis is the process used to determine and estimate risks – this may involve the definition and analysis of different loading conditions, failure modes and consequences as well as probability estimation (Cyganiewicz and Smart 2000). Risk assessment is the use of information from risk analysis to evaluate the various sources of risk and make decisions (Mcgrath 2000). This section focuses on techniques used for risk analysis.

There are a variety of different practices used in the analysis of risk and safety of engineered systems. Society of Automotive Engineers (SAE)'s Aerospace Recommended Practice document (SAE 1996), and a number of standards (IEC 2008, 2010) from the International Electrotechnical Commission (IEC) provide useful reference material for developing an initial understanding of the various assessment tools. Many of the approaches described by SAE and IEC are not mutually exclusive; that is, multiple approaches may be used in a system safety assessment, and the results of one approach may become the inputs of another. Four of the most commonly used techniques for system safety assessment, in particular within the dams industry, are Failure Modes and Effects Analysis (FMEA) and it's descendant Potential Failure Modes Analysis (PFMA), Fault Tree Analysis (FTA) and Event Tree Analysis (ETA). These are described in the following sections, which present the general theory, applications and limitations of each approach. Additional methods are then briefly described.

2.2.1 Failure Modes and Effects Analysis (FMEA)

FMEA is a systematic assessment approach that seeks to determine potential failure modes and identify their causes and the potential effects on system performance (IEC 2008). It was first developed in 1949 by the U.S. Military for weapons systems and refined in the 1960's for applications in the aerospace industry (Stamatis 2002; Thomas 2012). The use of FMEA was extended in the 1970's to automotive, aerospace and petrochemical industries (SAE 1967; National Research Council 1981) and was later applied in the

nuclear, food, drugs, and cosmetics industries (Duckworth and Moore 2010) as well as the dams industry (Hartford and Baecher 2004; dos Santos et al. 2012).

There are a number of different implementations of FMEA, however the general approach remains consistent (Thomas 2013). FMEA essentially identifies components and their failure modes, and then identifies the causes and effects for each failure mode. Failure Modes, Effects and Criticality Analysis (FMECA) adds an additional step where the severity and probability of events are used to determine the failure modes criticality (Thomas 2013). Details regarding the FMEA and FMECA processes are described in the IEC's International Standard 60812 (IEC 2008) as well as SAE's Aerospace Recommended Practice manual, ARP4761 (SAE 1996).

The IEC standard provides an overview of the information that should be made available to the team performing the analysis. In particular, the system boundary should be clearly defined and its elements, their characteristics, function and connections with other elements should be known. Levels of redundancy, system inputs and outputs, and information regarding how the system structure changes in response to different operating modes are also essential for the analyst team (IEC 2008). Representing the hierarchical system structure through the use of diagrams is recommended to illustrate relationships between components, redundancies and the inputs and outputs (IEC 2008). Information relating to maintenance routines, frequency of use of the different aspects of the system as well as operation should also be made available to the analysis team. The FMEA process described in IEC 60812 is illustrated in the following diagram (Schmittner et al. 2014):

The process involves identifying failure modes for a particular component, and then for each failure mode determining the effects, severity, causes and the frequency or probability. Analysis of severity may be done using qualitative descriptors such as catastrophic, critical, marginal and insignificant. This process is done for all components of the system at the particular level of detail of analysis. Once this detailed, component-level assessment is complete, the effects of failures on the next level of the system should be determined (IEC 2008). In a hierarchical system, the effects at the immediate level become the failure modes at the next level, and this can continue until the highest level of

the system is analyzed (IEC 2008; dos Santos et al. 2012). In this way, the immediate effects of a failure on the system as a whole can be determined. In FMECA, criticality is a qualitative measure of the relative degree of importance of a failure mode, and it is determined using the likelihood and severity of the failure mode (IEC 2008). There are a number of different ways in which criticality can be assessed and these are outlined in IEC 60812.

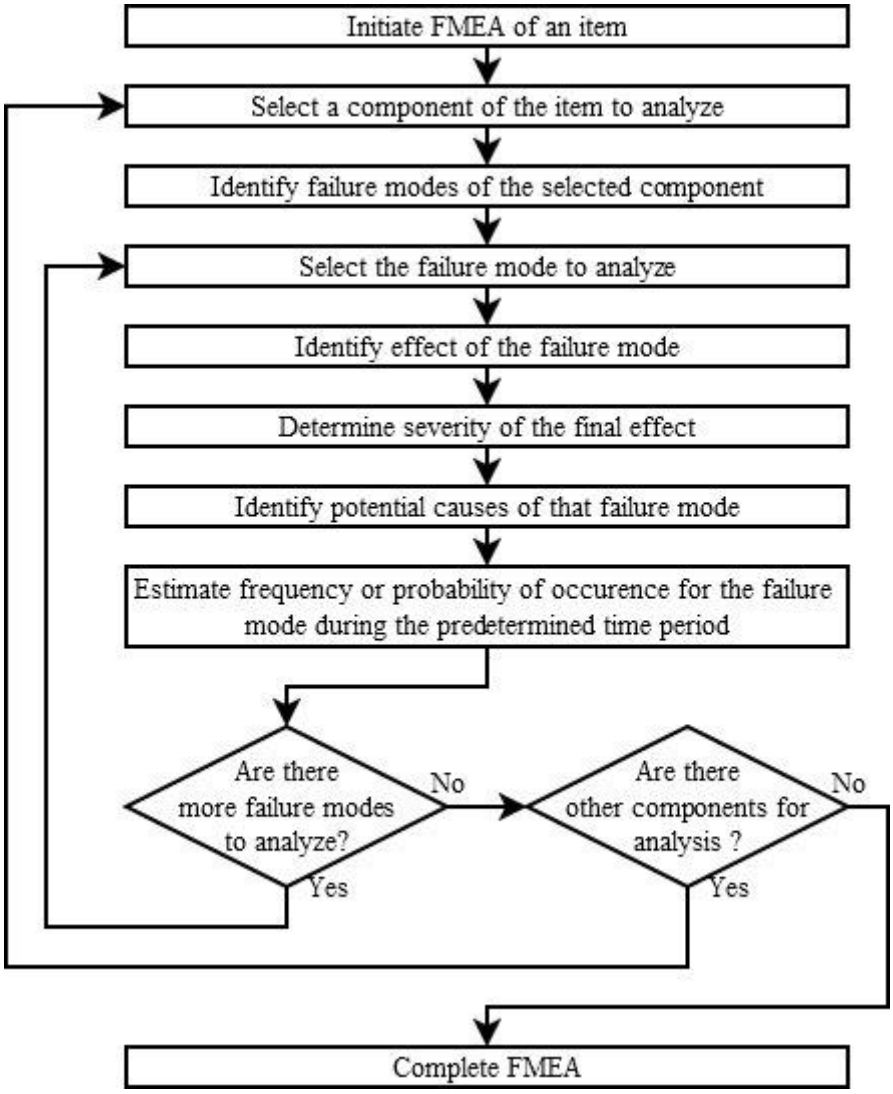


Figure 2-1: FMEA Process (Schmittner, 2014)

The FMEA/FMECA process typically involves a multi-disciplinary team of experts who work together to analyze the system (Hartford and Baecher 2004; dos Santos et al. 2012).

Information from the process is usually recorded in a tabular worksheet form and often uses the aid of diagrams and flow charts to illustrate how the event propagates through the system (Mcgrath 2000; Bartsch 2004; IEC 2008; dos Santos et al. 2012). The IEC standard (IEC 2008) and the dam-specific risk assessment text by Hartford and Baecher (2004) provide example worksheets similar to the one shown in Table 2-1. Analysts may wish to consider the ways in which a failure may be detected or prevented in the analysis and can use the results of the analysis to form conclusions regarding actions that can be taken to mitigate or eliminate important failure modes.

Table 2-1: FMEA Sample Worksheet

Component	Function	Failure Modes and causes	Local consequence	Global consequence	Ability to detect	Severity	Probability	Treatment action

Within the dams industry, there are some examples of FMEA being utilized for assessments of dam safety (Putcha and Patev 2000; Shaw et al. 2000; Hartford and Baecher 2004; dos Santos et al. 2012). Hartford (2001) notes that BC Hydro considered FMEA to be an important precursor to quantitative risk assessment as early as the mid 1990s. By the early 2000's the majority of American dam associations (eg. USBR, USACE, FMEA) began advocating for a heuristic FMEA-inspired approach called PFMA (see Section 2.2.2) that would reduce the time and effort required to complete a true FMEA.

The IEC and SAE standards note some limitations of the FMEA/FMECA approach. Despite successful applications in multi-level hierarchical systems, IEC (2008) states that analysis of multi-level systems can introduce complications and errors, suggesting that limiting the analysis to two levels of a hierarchical system is preferable. It is noted that the key assumption in FMEA is that failure modes are independent. The ability of the FMEA process to deal with common-cause failure is quite limited, and at best they can only be analyzed qualitatively (SAE 1996; IEC 2008). This means that only a subset of all possible failure scenarios are considered (Thomas 2013). There are also limitations in dealing with human factors and software errors that may contribute to system failures (IEC 2008). Nonlinear and feedback relationships are unable to be analyzed effectively using FMEA/FMECA (Thomas 2013), so failure initiation and progression can be extremely

difficult to assess (Shaw et al. 2000; Bartsch 2004). Zhang et al. (2018) note that analysis of redundancy in systems is complicated by component interdependency. The IEC (2008) suggests utilizing fault tree analysis to deal with interrelationship scenarios and common cause failures. Thomas (2013) suggests that FMEA/FMECA, by its nature, is only able to analyze scenarios that are triggered by a failure – the result is a set of both safe and unsafe scenarios, with an equal amount of time spent analyzing each. There are, however, some unsafe scenarios which may not be triggered by failures at all, and these are omitted from the analysis. Dos Santos (2012) and Zhang et al. (2018) suggest that components may take on multiple potential states so the binary definition of functional or failed may not be adequate. Many authors have also noted that FMEA is a time and resource consuming process which requires a significant amount of information and spends considerable time analyzing less-relevant failure modes (Mcgrath 2000; Shaw et al. 2000; Bartsch 2004; IEC 2008; dos Santos et al. 2012). Shaw et al. (2000) points out that FMEA was developed for active systems, in which each component has an output action – whereas in dam systems, many of the components are passive. dos Santos et al. (2012) found that FMEA did not give adequate consideration to time dependencies or deterioration where in reality, some components progress slowly towards a failed state.

The advantages and disadvantages of the approach are summarized in Table 2-2. In spite of these limitations, FMEA/FMECA still provides a useful and systematic tool for the identification and assessment of potential failures modes in a variety of systems and processes. Identifying failure modes is an important aspect of risk analysis and for the analysis of system safety in general. Knowing what the possible failures are facilitates the development of strategies to eliminate, detect, mitigate and/or reduce their likelihood of occurrence.

Table 2-2: Advantages and disadvantages of FMEA

Advantages	Disadvantages
-Systematic approach[1,2,3] -Determines potential failure states of system components and their effects on other components of the system[1,2,3]	-Failure-focus that can miss unsafe non-failure component states or interactions[2] -Difficulty in analyzing redundancy[1,3] -Static analysis with limited ability to analyze feedback, interaction, time, dynamic system behaviour[1,2] -Common-cause failures, human factors and software errors are challenging within this framework[2] -Difficulty and significant complexity with multi-level hierarchical system analysis[1,3]

1 Hartford and Baecher (2004)

2 Thomas (2013)

3 IEC (2008)

2.2.2 Potential Failure Modes Analysis (PFMA)

PFMA is a qualitative analysis tool that is utilized primarily within the dams industry. PFMA is essentially a simplified, heuristic variant of Failure Modes and Effects Analysis (FMEA) which was developed in the early 2000's by FERC in response to the time and resource commitments required to perform a comprehensive FMEA (Hartford and Baecher 2004; France et al. 2018a). Dam safety literature pre-dating FERC's introduction of PFMA (FERC 2005b) often refers to FMEA (eg. Bowles et al. 1998b; Putcha and Patev 2000; Stewart 2000; Barker et al. 2003; Faber and Stewart 2003; Hartford and Baecher 2004; Wieland et al. 2005), though many researchers and dam agencies now use the simplified PFMA methodology as a result of the influence of the American dam associations (eg. Bowles et al. 2011; USACE 2011; SPANCOLD 2012; USBR and USACE 2012b; Adamo et al. 2017).

Despite sometimes being mentioned as a single approach, there are fundamental differences between PFMA and FMEA/FMECA. FMEA/FMECA is a systematic approach that works up from the most detailed level of a hierarchical system to higher levels of the system. In PFMA, this is done heuristically for the system as a whole. This also means the definition of a failure mode may be slightly different between the two methods. In FMEA, a failure is defined as a components ability to achieve it's intended function – failures are defined at the component level. In PFMA, failures are defined at the system level – only events which result in a problem at the system level are considered.

PFMA is essentially a failure mode brainstorming session involving a team of experts, including engineers, field staff and operating staff. The PFMA team performs a review of all existing data, historical records and information and uses this to come up with possible modes of failure for the dam, including their causes, qualitative likelihood descriptors, and consequences (FERC 2005b). The failure modes are categorized into the following groups (FERC 2005b):

- Category I – Highlighted potential failure mode (increased significance and likelihood)
- Category II – Potential failure modes considered but not highlighted (lesser significance and likelihood)
- Category III – More information or analysis needed to classify
- Category IV – Potential failure mode ruled out (physically impossible or unlikely)

FERC (2005b) states that the result of a PFMA analysis is an information resource that can help illuminate failure modes not previously considered, while highlighting the importance of failure modes with high consequence and likelihood. The guidance document also states that the analysis may identify some failure modes which are less significant than previously thought, due to their associated consequences or likelihood (FERC 2005b). It is suggested by some researchers and organizations (eg. Bowles et al. 1999, 2011; FERC 2007; USBR and USACE 2015; Adamo et al. 2017) that PFMA is the first step in risk assessment, to be followed by the quantification of risk, using event trees or other guidelines.

In 2003, The Federal Energy Regulatory Commission (FERC) made PFMA a requirement for all American dams meeting certain criteria (relating to hazard level and size under the Code of Federal Regulations 18 Part 12 Subpart D), and published a technical document

detailing the steps of the analysis (FERC 2005b; Hoeg et al. 2007). The USACE and USBR also include PFMA in their dam safety policies and best practices manuals (USACE 2011; USBR and USACE 2015b). Several other agencies have their own guidelines, webinars and supporting documents (Hydrometrics Inc. 2011; USSD 2013; ASDSO 2018). The CDA dam safety guidelines do not specifically mention PFMA (CDA 2007).

Despite its widespread use and successes in identifying some failure modes as well as helping dam owners prioritize risk reduction measures (eg. FERC 2007; Adamo et al. 2017), the PFMA process is not without its shortcomings. This became especially apparent following the Oroville Dam incident of 2017 (France et al. 2018a).

In the aftermath of the incident, an independent forensic team consisting of engineers from a variety of organizations was assembled to review the causes of the incident (France et al. 2018a). It was revealed that the dam had been the subject of three PFMA's in 2005, 2009 and most recently in 2014. Failure modes relating to the emergency overflow spillway and spillway chute were overlooked in 2005 and 2009. In 2014, two relevant failure modes were identified. The first was related to the emergency spillway: "A PMF flood event is occurring and over 10 feet of water is spilling over the emergency spillway at Oroville Dam. Erosion begins where the flow is entering the Feather River and progresses by head-cutting into the reservoir" (France et al. 2018a). The possibility of erosion happening at lower spillway flows was not considered and the failure mode was classified as Category IV (non-credible) as a result of its perceived likelihood being small. The second relevant potential failure mode was the failure of the spillway chute: "Cavitation or slabjacking results in loss of the concrete lining in the spillway chute downstream of the [spillway gates]. The rock in the spillway chute erodes and the [spillway gates are] undermined and lost" (France et al. 2018a). Again, this potential failure mode was classified as Category IV (non-credible), as a result of its perceived likelihood being small.

The Oroville incident has helped highlight some of the shortcomings of PFMA. In two of the three PFMA sessions, the pertinent failure modes were completely missed, and in the third, their likelihoods were perceived to be so low as to not warrant further investigation or remediation. The 2014 PFMA was the result of two weeks of analysis and was

considered to be very thorough by all involved, however some key errors in judgement were made relating to assumptions about the geologic conditions and the condition of the spillway chute (France et al. 2018a). France et al. (2018a) notes that a considerable emphasis is placed on loss-of-life during the PFMA process, resulting in some plausible scenarios being eliminated from the list and categorized as Category IV due to perceived minimal consequences. In addition to this, there tends to be more emphasis on the dams themselves, and less emphasis on their appurtenant structures. France et al. (2018a) also noted issues with the FERC categorization scheme – FERC (2005b) states that "If you do not fully develop a [potential failure mode], you cannot categorize it", when in reality more information is often necessary to properly categorize some of the potential failure modes. Another key issue with the PFMA process is that the results are subjective and could vary significantly depending on the individuals involved, the data available and the time allocated for the process. This is obvious when considering that two of the three PFMA reports overlooked failure modes relating to the spillways (France et al. 2018a). The independent forensic investigators also noted that PFMA was considerably less structured than FMEA.

The advantages and disadvantages of PFMA are summarized in Table 2-3.While PFMA represents a positive step by the dam industry towards considering more than just engineering standards like the PMF and MDE, there are some limitations to simplifying the analysis of such complex systems. The interactions among components and the consequences of these interactions may be missed in the analysis. The system is not necessarily broken down and analyzed in a hierarchical way as it is in FMEA/FMECA, so lower level failures may be overlooked. The approach suffers from the same issues as FMEA/FMECA in terms of common cause failures, human factors and software errors. The classification scheme allows practitioners to "rule out" lower-consequence or less likely events, despite the fact that these events do contribute to the overall system risk. The focus of the PFMA reports available publicly tends to be on dam breach or collapse over other seemingly less consequential modes of failure. Many researchers and agencies (eg. Bowles et al. 1999, 2011; FERC 2007; SPANCOLD 2012; USBR and USACE 2015; Adamo et al. 2017) consider PFMA to be the first step in quantitative risk assessment (event tree analysis is often recommended as the next step). While identification of failures modes

is an important first step in quantitative risk assessment, PFMA effectively rules out some failure modes and could potentially be missing others. It is also not possible using PFMA to systematically identify possible sequences of events which may be benign on their own, but together could lead to catastrophic consequences (Hartford et al. 2016).

Table 2-3: Advantages and disadvantages of PFMA

Advantages	Disadvantages
-Improves the understanding of dam risks by determining the site-specific causes of potential issues[1] -Efficient[1]	-Heuristic brainstorming approach that relies on expert judgement and mental models of complex systems[2] -Failure-focus that can miss unsafe non-failure component states or interactions -Tendency to analyze only conditions that lead to uncontrolled release of the reservoir[2] -Static analysis of linear chains of events, with limited ability to analyze feedback, interaction, time, or dynamic system behaviour[2,3] -Human and operational aspects of failures difficult to analyze[2] -Some failure modes may be determined "non-credible" but may still have significant safety impacts[2]

1 FERC (2005b)

2 France et al. (2018a)

3 Regan (2010)

2.2.3 Fault Tree Analysis

Fault trees were originally developed in the 1960's to analyze missile systems (Thomas 2012). Fault Tree Analysis (FTA) was developed as a way of identifying combinations of

failures and determining their likelihoods. They were first applied to assess the launch system of Minuteman I, and were extended for components throughout the system on Minuteman II (Ericson 1999). Boeing began using fault trees to assess aircraft by 1966 and by the end of the 1960's, fault trees were the standard of practice for the weapons and aerospace industries (Ericson 1999). In the early 1970's, fault trees were adopted by the nuclear and chemical industries and software systems were developed to improve analysis abilities (Ericson 1999; Thomas 2012). In the 1980's and 1990's, the approach was being applied in the robotics industry (Lin and Wang 1997) and software industry (Leveson 1995; Hansen et al. 1998).

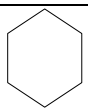
Fault trees can be thought of as the mirror image of event trees, in that they use deductive reasoning whereas event trees use inductive reasoning. Fault trees start with an undesirable event and proceed from the general to the specific, using a backward logic to determine the potential causes of an undesirable event (Hartford and Baecher 2004). The result is a graphical depiction which moves down the page in levels of detail that progress with each step in the tree (SAE 1996). This results in a tree structure which shows how combinations of undesirable events or failures at lower level components can cause the event in question. In fault trees, faults are the undesirable events (also known as "top events" and lower level events are failures (Thomas 2012). In FTA, a separate fault tree would be constructed for each undesirable event (SAE 1996). There are several graphical constructs used in the development of fault trees including what are known as "logic gates", which are presented in Table 2-4 (Lee et al. 1985; SAE 1996). There are also graphical constructs showing events, with different symbols representing different types of faults (Lee et al. 1985).

The development of a fault tree starts by determining what immediate failures would be responsible for the top event (fault). The analyst then moves down the tree in increasing levels of detail, determining the causes of each failure and linking them using the appropriate logic gates. Fault tree creation stops when the root causes of an event are determined or further development is deemed unnecessary (SAE 1996).

The act of constructing the fault tree alone can provide useful information in terms of what needs to fail for the top event to occur. A cut set is a unique combination of events within

the fault tree that could lead to failure. A fault tree may have many cut sets depending on its level of complexity. The smallest set of events that can lead to the top event is known as the minimal cut set (Thomas 2013). Investigating the cut sets and organizing them based on the number of primary events in each cut set can be a useful qualitative tool for determining what primary events (or combinations of them) are the most concerning.

Table 2-4: Basic event tree logic gates

Symbol	Name	Description
	AND	TRUE if all input events occur
	OR	TRUE if at least one of the input events occurs
	VOTE	TRUE if at least m of the input events occurs
	EXCLUSIVE OR	TRUE if only one of the input events occurs
	PRIORITY AND	TRUE if input events occur in a particular order
	INHIBIT	TRUE if all inputs event occur, as well as an additional (typically external) event

Determining the minimal cut set can be challenging if events occur in multiple places within the fault tree – this can happen if there is dependence between two or more events and is relatively common for complex systems (SAE 1996). For fault trees without interdependence (each event occurs only once), relatively simple rules can be followed to determine the probability of the top event. Assuming the probability of an event A can be represented by P(A), the basic rules for quantitative analysis are derived from set theory and are determined as follows (SAE 1996):

- The probability of events A AND B both occurring is $P(AB)$ and is equal to $P(A) * P(B)$. For three events connected by AND gates, the three probabilities are

multiplied, and so on. This represents the intersection of sets *A* AND *B*, as shown in Figure 2-2 by the area represented by AB.

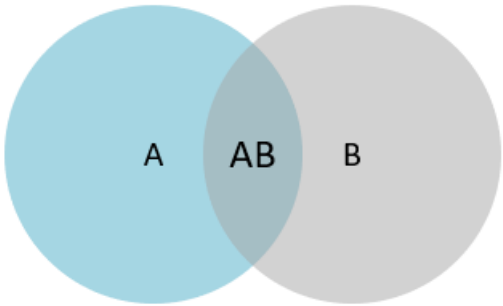


Figure 2-2: Illustration of the intersection of A AND B

- The probability of events *A* OR *B* occurring can be denoted $P(A + B)$ and is equal to $P(A) + P(B) - [P(A) * P(B)]$. For three events connected by OR gates, the equation becomes $P(A + B + C) = P(A) + P(B) + P(C) - [P(A) * P(B)] - [P(B) * P(C)] - [P(C) * P(A)] + [P(A) * P(B) * P(C)]$. The set theory used to develop the second equation is shown in Figure 2-3. The total area shaded is known as the union of sets A, B, and C. Each of the two-circle intersections (AB, AC, BC) is negated once to avoid double counting, and then the three-circle intersection (ABC) is re-added to ensure the complete area is counted for.

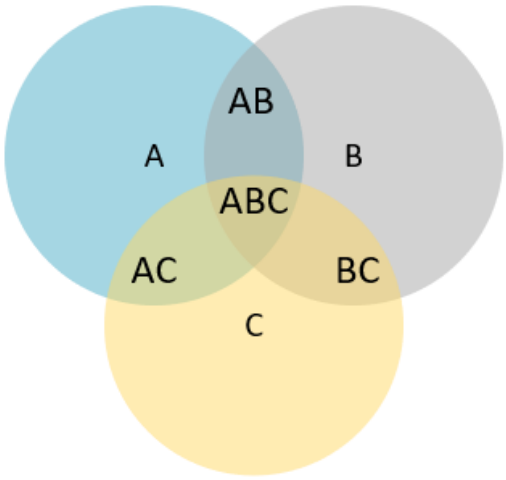


Figure 2-3: illustration of the union of sets A, B and C (OR)

- The probability of mutually exclusive events occurring (if one occurs the other can't) is simply equal to $P(A+B) = P(A) + P(B)$, with $P(AB) = 0$. The set theory used to derive this concept is shown in Figure 2-4. Note in this illustration, there is no overlap between sets A and B, which indicates mutual exclusivity.

Figure 2-4: Illustration of mutual exclusivity in sets A and B

These basic rules, derived from simple set theory, can be applied to calculate the probability of occurrence for the top event. This is illustrated using an example shown in Figure 2-5. In Figure 19, the top event can only occur if both Failure A AND Failure B occur. Failure A can only occur if one OR more of Failure C, D, or E occur. The basic probabilistic analysis process is shown in the example figure. To calculate the probability of the top event, the probabilities of Failures B, C, D and E are required. The possible cut sets are CB, DB, EB, CDB, DEB, CEB, and CDEB. Failure A has its own set of primary events, so it is not a primary event and is not counted in the cut sets, while Failure B is a primary event because it is not further decomposed.

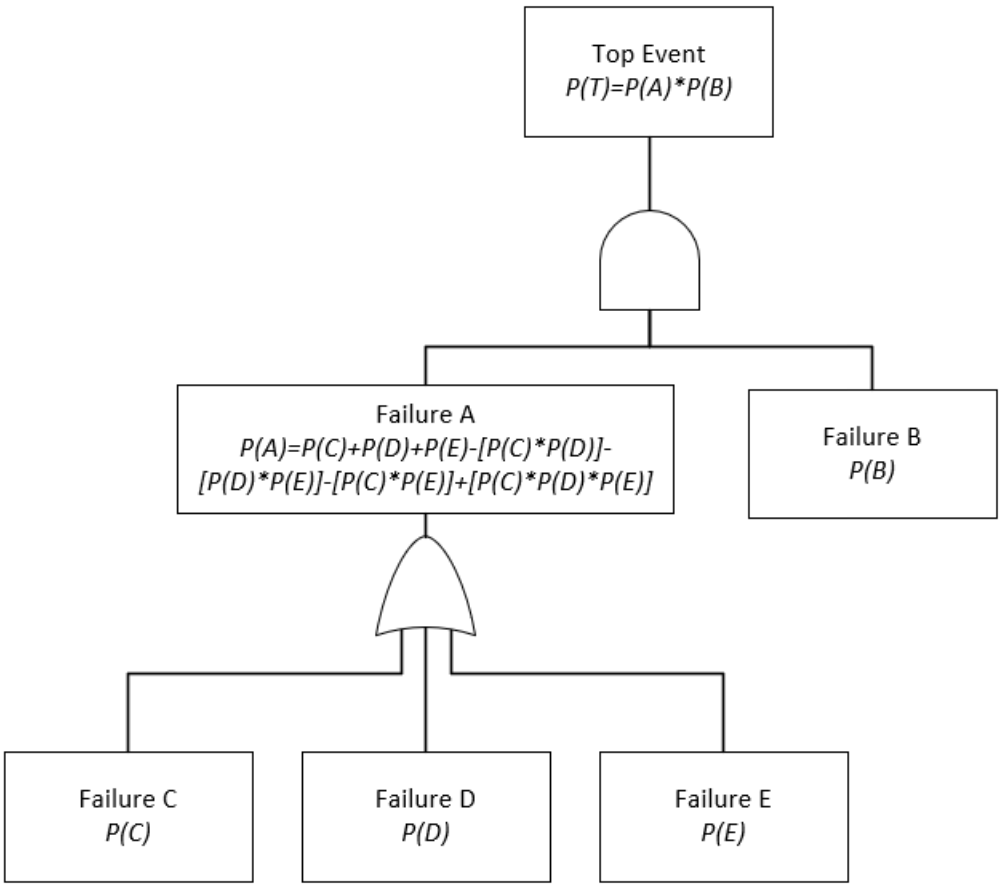


Figure 2-5: Simple fault tree example

As mentioned previously, fault tree analysis can be complicated when events or failures appear multiple times within the tree. This indicates common-cause failures (SAE 1996). Qualitatively, this can be analyzed by looking at the list of possible cut sets. For cut sets where a failure can lead to more than one primary event, a common cause failure has occurred. This more complex situation necessitates the use of Boolean Analysis to appropriately compute the probability of the top event (SAE 1996). Boolean algebra is a mathematical formulation that deals with True or False events (1's and 0's), which is useful for fault trees where events either occur or do not occur (Hartford and Baecher 2004). The SAE Aerospace Recommended Practice manual provides an excellent example of "Boolean reduction" in analysis of complex fault trees (SAE 1996). The method for determining the minimal cut sets is shown and the Boolean Logic rules are described. There are a variety of computational software packages available to aid in the construction and

mathematical assessment of fault trees, including Prepp/Kitt, SETS, FTAP, Importance, Isograph and COMCAN (Lee et al. 1985; Ericson 1999; Barker et al. 2006). These tools are typically capable of performing cutset determination and probability calculation automatically (SAE 1996).

Due to its general simplicity and ability to graphically illustrate potential paths to failure, FTA has become widely used across a variety of industries as mentioned previously. It is a powerful tool that provides excellent insight into the possible ways in which primary events (failures) can lead to top events. The limitations of the approach are also well documented. Thomas (2012) notes that the approach relies heavily on the initial list of top events – it does not determine these. Some other approach must be used. FTA also heavily relies on the quality of information pertaining to the system of interest, and it may be possible to omit events from the tree inadvertently. No systematic techniques are available to ensure an exhaustive analysis is completed. Thomas (2012) notes that the decomposition in an FTA often stops at a subjective point in the analysis, where causes of failure become less obvious and more complex – essentially, "FTA often finds what is only intuitively obvious". Human factors and software errors are not easily reduced to a simple binary representation (failing or functioning) and as such they are not easily incorporated into an FTA analysis (Thomas 2012). Perhaps most importantly, FTA focuses on failures alone, and as such it may omit non-failure causes of a top event that could occur as a result of design errors, omissions or other factors such as delays and human error. Many issues that contribute to accidents historically are dynamic processes that may not be easily represented as simple failures (Thomas 2012). Furthermore, the fault tree assumes linear relationships among system components and is unable to capture component interactions or time-dependent, nonlinear feedback behaviour which may lead to unexpected outcomes for the system of interest.

Hartford and Baecher (2004) provide an excellent overview of FTA within the context of dam safety. They describe methods of estimating the probability of the top event when common-cause failures are observed. Their review of FTA also discusses methods for determining the relative importance of cutsets. In general, there are a limited number of dam safety applications of FTA available within the public domain. Putcha and Patev

(2005) describe the application of fault trees as a method for analyzing dam gates and operating equipment, presenting generalized fault trees showing how a gate may fail to open or close. These generalized fault trees could be useful for practitioners as a starting point for application to a specific system of interest. Putcha and Patev (2005) suggest the use of criticality indices derived through FMEA as a means of ranking the relative importance of components within the gate system fault trees. They go on to use the criticality along with failure rates to determine probabilities of individual component failures, which can then be used to determine the overall probability of the top event (Patev et al. 2005). The approach presented provides a good starting point for fault tree analysis of dam gate systems.

Barker et al. (2006) used fault tree analysis to assess the reliability of various options for a spillway system upgrade in Queensland, Australia. A variety of different operating states were analyzed using fault trees, and human error was included in the analysis. Several scenarios were tested, and sensitivity analysis was performed for various assumptions regarding grid reliability, operating staff assumptions, PLC reliability and redundancy, as well as backup power source reliability and redundancy (Barker et al. 2006). The authors note that the results of the analysis were useful in selecting the final configuration for the system upgrade, but do not show the fault tree arrangement or mathematical computations used.

The advantages and disadvantages of FTA are summarized in Table 2-5. Hartford and Baecher (2004) note some disadvantages to the use of fault trees for analysis of dam systems, mainly pertaining to the high level of complexity in large fault trees and the reliance on expert judgement in their construction. Nevertheless, they note that fault tree analysis may be the only alternative in some cases to modelling complex systems in an attempt to understand and quantify failure modes. One key issue in the use of fault trees is they represent a linear event progression (Thomas 2012). In dam systems, components may not instantaneously progress towards a failed state, instead degrading in some way over time. Inflows introduce another nonlinear variable that complicates analysis using fault trees. Traditional FTA may not be capable of characterizing the reservoir level with respect

to different system operating conditions and inputs, which is an important goal in the context of dam safety.

In more recent studies, Bayesian Networks and algorithms have been applied as a means to overcome some of the limitations associated with fault trees (Ching and Leu 2009; Jong and Leu 2013). Ching and Leu (2009) used a Poisson process to model time-varying arrivals of disturbances, representing the system using a fault tree model with a Bayesian algorithm incorporated to assess uncertainty. The goal of the analysis was to model how the reliability of civil infrastructure changes over time. The approach was demonstrated on a spillway gate system for a dam in Taiwan and was found to offer a fast solution that helped overcome some of the issues associated with lack of failure rate data. Results showing remaining life and failure rate plotted against time are shown for the case study. Jong and Leu (2013) applied a hybrid approach using fault tree analysis in conjunction with Bayesian Networking to overcome some of the limitations associated with both approaches. Their approach was to transform fault trees, which are more easily and logically developed, into Bayesian Networks, which are more tedious and difficult to set up for complex systems but allow for expert knowledge to be incorporated with Bayesian Probability Theory for improved diagnosis of system faults (Jong and Leu 2013). The approach was demonstrated on three Taiwanese dam systems and shown to match Weibull-distribution based reliability analysis of those systems. While these approaches do address the traditional FTA limitation of failure rates that change over time, they do not consider system inflows, interactions between components or the overall system response to component failures. These issues remain outside of the capabilities of FTA at the current time.

Table 2-5: Advantages and disadvantages of FTA

Advantages	Disadvantages
-Logical and visual method for displaying failure paths through a system[1] -Can be used to estimate probability of top events and unique paths to the top event[2] -Works very well at identifying the importance of component failure modes[3]	-Failure-based method that can miss unsafe scenarios caused by interactions or non-failures[3] -May not follow system flow diagram so it can be difficult to relate fault tree logic to the actual interactions within the system[1] -Difficulty capturing software errors or human behaviour[3] -Relies on mental models of system structure and expert judgement[3] -Static analysis with limited ability to analyze time or dynamic system behaviour[3] -Discrete component states for variables that may be continuous or have multiple states[3]

1 Hartford and Baecher (2004)

2 Lee et al. (1985)

3 Thomas (2013)

2.2.4 Event Tree Analysis

Event Tree Analysis (ETA) was originally developed for safety assessments of nuclear power plants in the United States in 1975 through the WASH-1400 study (IEC 2010; Thomas 2013). The original goal of the WASH-1400 study had been to develop a large and detailed fault tree of the system, but it was determined that this would be far too large to be practical. Event trees were conceived as a means of defining potential accident paths, where failures within the path could be further deconstructed using FTA (Thomas 2013).

Despite being developed for use alongside FTA, ETA has also been presented as a separate tool for analysis of system dependability (Skelton 1997; Rausand and Hoyland 2004; IEC 2010).

Rausand and Hoyland (2004) define ETA as an inductive technique that begins with a problem in the system (an initiating event), and proceeds to identify paths by which the problem may develop. ETA is similar to FTA in that it is a chain-of-events type analysis, but differs from FTA in that it starts with an event and proceeds forward to determine the possible outcomes (Thomas 2013). Event trees can be used to determine the probability of the possible outcomes resulting from an initiating event (IEC 2010). The International Electrotechnical Commission has published a standard detailing ETA which documents the steps in event tree development and quantitative assessment of outcome probabilities (IEC 2010). In ETA, mitigating factors are considered to be factors within the system that are intended to reduce the consequences of an initiating event. ETA then logically steps through each of these mitigating factors and determines what happens next when the factor either succeeds or fails to perform its intended function (Rausand and Hoyland 2004; IEC 2010). The different steps of the event tree are called nodes, and their probability can be calculated using FTA, as originally intended by the developers in the WASH-1400 study (IEC 2010). The probabilities of each unique path in the event tree are then simply multiplied together to estimate the ultimate probability of the outcome.

It is important to note that in an ETA, the initiating events are not determined through the analysis. Rather, the sequences of events and outcomes that could possibly result from an initiating event are determined and their probabilities are quantified (IEC 2010). In this way, ETA is not a standalone analysis tool (Thomas 2013). Initiating events may be determined using some other form of analysis. Rausand and Hoyland (2004) mention FMECA along with Preliminary Hazard Analysis (PHA) and hazard and operability analysis (HAZOP) as potential techniques to determine the initiating event.

The first step in an ETA involves clearly defining the system of interest and its boundaries. Next, initiating events are selected and the mitigating factors required to prevent outcomes or accidents are determined and organized depending on their respective time of

intervention (IEC 2010). The success or failure of a mitigating factor determines the next step in the tree and in this way, sequences of events are defined. Each unique path through the tree represents a unique sequence of events. The probability of success or failure of each mitigating factor is multiplied together along with the initiating event's probability, $P(I_E)$ to determine the probability of the outcome (Rausand and Hoyland 2004). This is demonstrated using the simple example shown in Figure 2-6.

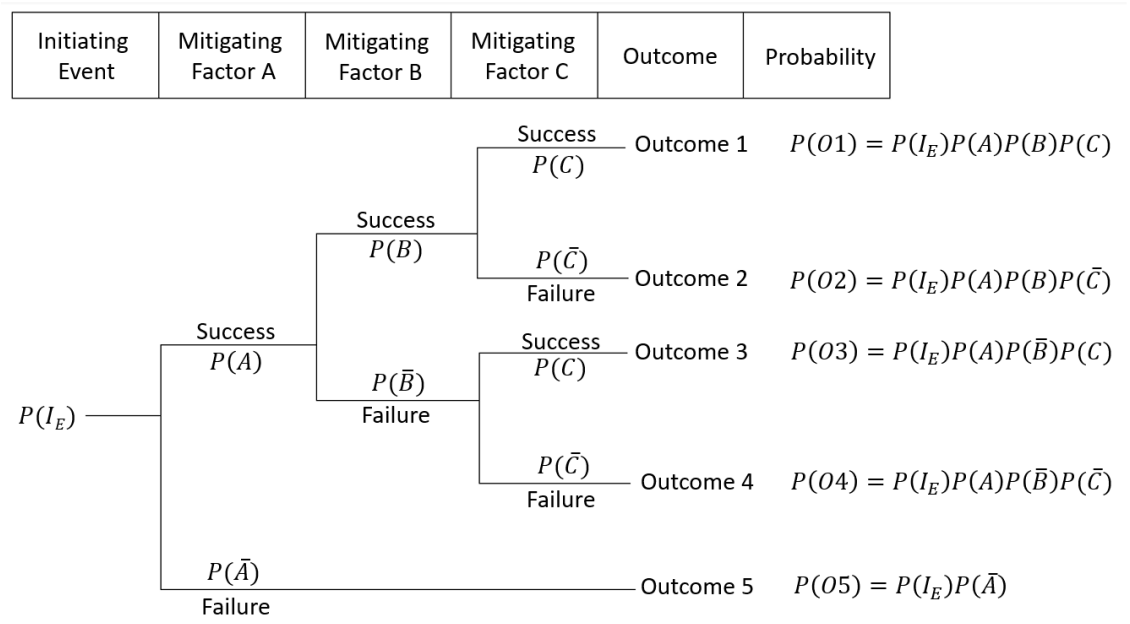


Figure 2-6: Generic event tree with probability calculation

The IEC standard specifies that overbars are used to indicate failed mitigating factors (IEC 2010). For example, the probability of mitigating factor A failing is $P(\bar{A})$. The probability of success and failure are mutually exclusive. That is, $P(A) = 1 - P(\bar{A})$. A success and a failure may not occur concurrently.

In general, the ETA method is relatively easy to apply, and quantification uses straightforward mathematical concepts. It is a useful tool for visualization of event chains and can enable identification of outcomes that may not be generated using simple brainstorming (IEC 2010). With a complete set of initiating events, ETA provides a useful tool for depicting and analyzing potential system outcomes. Event trees are capable of improving the understanding of various failure modes and estimating the likelihoods of

failure of systems in general. They remain a widely used tool across many industries, including but not limited to hazardous processes (Ferdous et al. 2011; Villa and Cozzani 2016), supply chain risk management (Tummala and Schoenherr 2011), infrastructure risk management (Ezell et al. 2000), and nuclear safety (Rychkov and Kawahara 2015). Many of the recent applications of ETA incorporate new techniques to deal with uncertainty in the probability estimates (Ferdous et al. 2011; Srivastava 2013) or dynamic behaviour (Bowles et al. 2011b; Rychkov and Kawahara 2015).

Like all approaches, ETA is not without its limitations, and these are well documented within the applicable reference material. The approach is inherently reliant on practical experience and understanding of the system (IEC 2010). Thomas (2013) points out that because ETA starts with an initiating event and the functions in place to mitigate its consequences, preventative measures for the event itself can not be included in the analysis. Software has also been developed to address human factor considerations in event trees, however the decisions are typically either randomized or reduced to binary variables of success or failure (Thomas 2012). Thomas (2012) suggests that the mitigating factors in event trees are assumed to be independent of one another, when in reality this may not be the case. He cites the Three Mile Island and Fukushima nuclear incidents as an example of how seemingly independent issues may be caused by the same factors (Thomas 2013). Multi-state variables are unable to be modelled in event trees, despite being present in may complex systems (Villa and Cozzani 2016). Finally, Thomas (2013) states that ETA is fundamentally a failure-based method focusing on the propagation of component failures through the system. As such, an entire subset of potentially unsafe scenarios that do not involve failures at all may be impossible to assess through the use of ETA. Importantly, Hartford et al. (2016) suggests that the nature of ETA requires that events and their combinations must be identified at the outset. Because of this, the development of thorough and complete event trees is incredibly challenging since the number of physically possible conditions is so large. Further, consideration of time, feedbacks and nonlinear behaviour present additional challenges that are not obviously surmountable given the current state of the science (IEC 2010; Hartford et al. 2016).

Because of its relative simplicity for the analysis of complex systems, ETA has become a prominent tool in dam safety risk assessment since the mid 1980s (Whitman 1984; Stedinger et al. 1989; Bowles et al. 1999; Hill et al. 2001; Hill and Bowles 2003; Goodarzi 2010). They are considered by many to be the next step in quantitative risk assessment after a PFMA, in particular by the American dam associations (FERC 2005b; Bowles et al. 2011b; USACE 2011; USBR and USACE 2015b).

In the dams industry, the approach is commonly paired with PFMA to further analyze and quantify chains of events (USBR and USACE 2015b). PFMA is used to come up with the initiating events and ETA is used to quantify the various potential outcomes. Figure 2-7 (Hill et al. 2001) illustrates an event tree for a fictitious dam, showing how the event propagates through the system. Some event sequences are collapsed at the black nodes.

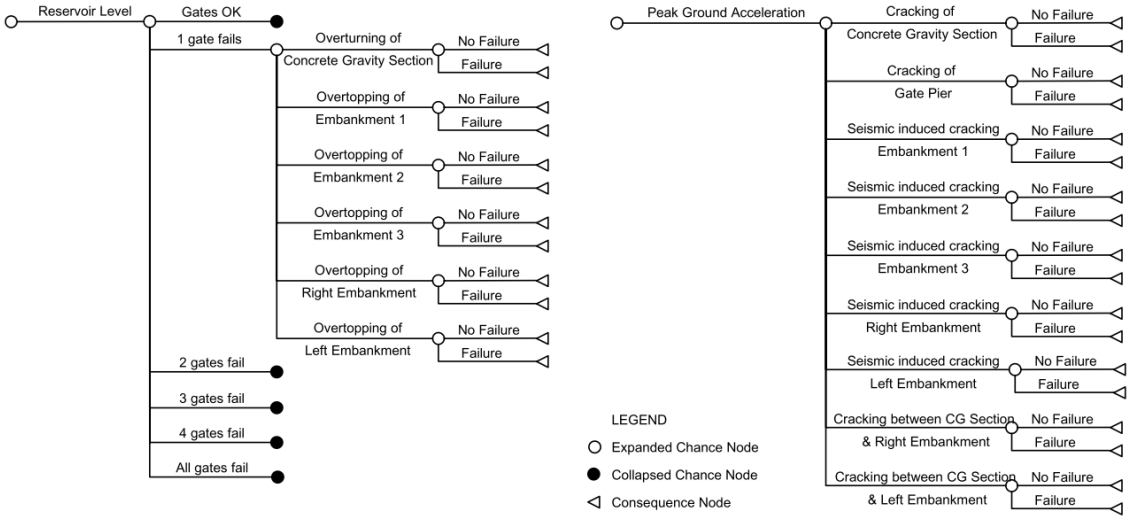


Figure 2-7: Event tree examples (Hill et al. 2001)

Whitman (1984) was one of the first authors to apply event trees in the dams domain, modelling the geotechnical aspects of dam safety with an event tree that progressed from embankment dam cracks through drain and filter states to either non-failure, piping failure or slope instability. The event tree is described as a very simplistic and generalized representation and interpretation of possible outcomes; however, it is thought by the author to give some structure to a process that would otherwise be very subjective. Similar, high level event tree examples from the early dam safety ETA literature are presented by Bowles

et al. (1987) and Yeigan (1991). Quantifying the probabilities of events in the event trees presents a major analytical challenge, so in early applications of risk analysis a verbal guideline or "Kent Chart" was used. Kent Charts were developed by Sherman Kent in the 1960's and were adopted by the CIA for a brief time to assign numerical probabilities to verbal descriptors (Hartford 2001). Vick (1992) presented one such chart for use within the dams domain.

By the mid 1990's and early 2000's, a "de-compositional" approach to event trees began being used. In response to the subjectivity of the simplified approaches being utilized, BC Hydro began investigating the use of analytical techniques for estimating probabilities in event trees (Hartford 2001). Hartford (2001) presented a detailed event tree as part of the quantitative risk assessment for seismic events at Hugh Keenleyside dam in BC, noting that this was the first analytically based risk analysis of a dam performed to date. Analytical and numerical techniques were used in the quantification of failure probabilities instead of subjective judgement. Another early application of quantitative, analytical risk assessment from the late 1990's was for the Hume Dam in Australia (McDonald and Wan 1999). By the late 1990's both BC Hydro and the Australian engineers who performed the Hume Dam assessment had concluded that simplified risk analysis, using Kent Charts and high-level event trees, was not sufficient to provide conclusive evidence of a dam's degree of safety or for use in dam safety decision making (Hartford 2001).

In more recent years, the use of event tree analysis has become more widespread and computational tools have been developed to improve the degree of analysis that can be achieved. DAMRAE is an event tree software which improves the capabilities of event trees, allowing for (a) modelling of continuous variables (such as inflow or ground acceleration) and (b) modelling of deterministic relationships between variables, for example the reservoir stage-discharge relationship or the deformation function as a result of earthquake loading and initial conditions (Srivastava 2008, 2013; Srivastava et al. 2012). The DAMRAE software was able to overcome some of the issues with the earlier applications of event trees and provides a path forward for use of this technique in the future. It was developed for the USACE to be used in their dam safety risk management program and is included in USACE's dam safety policy and procedures (Bowles et al.

2011a; USACE 2011). The DAMRAE software has since been applied in several applications (Bowles et al. 2010, 2011a, b, 2015). Srivastava (2013) includes a detailed description of DAMRAE and uses an example system to show how it can be used to test various risk-reduction alternatives.

A summary of the advantages and disadvantages of ETA is presented in Table 2-6. Regan (2010) has identified examples of dam failures, including Teton and Taum Sauk, in which nonlinear behavior was observed, noting that event trees are too simplistic to anticipate the complex interactions occurring within various levels of a dam system. This echoes the general conclusions made by Thomas (2012) with respect to accidents in the nuclear and aerospace domain. Zhang et al. (2011) note that ETA may not be suitable for analysis when there are multiple initiating events. Dam systems involve dynamic, interacting components with time-varying inputs. These result in a time-dependent evolution of events, which the IEC (2010) identifies as another limitation of ETA. The development of more advanced software overcomes some of these limitations, however it remains challenging to include timing in event tree analysis (Hartford et al. 2016). Because of this, ETA has limited applicability for dam safety applications in which reservoir response to disturbances occurring over time is a specific goal. Despite these limitations, it remains a useful tool for envisioning and understanding general possibilities for event propagation through complex systems.

Table 2-6: Advantages and disadvantages of ETA

Advantages	Disadvantages
-Logical and visual method for displaying sequences of events[1] -Very efficient way to estimate the probability of failure as a result of an initiating event[1]	-Failure-based method that can miss unsafe scenarios caused by interactions or non-failures[2] -Static analysis of a one-way chain-of-events with limited ability to analyze feedbacks, time or dynamic system behaviour[2] -Difficulty capturing software errors or human behaviour[2] -Discrete component states for variables that may be continuous or have multiple possible states[2] -Difficult to assess common-cause failures[2]

1 Hartford and Baecher (2004)

2 Thomas (2013)

2.2.5 Additional methods

The following sections provide a brief overview of some additional methods that may be used in system safety assessment.

2.2.5.1 Dependence Diagrams (DD)

Dependence Diagrams (DD) are described by SAE (1996) as “pictorial representation[s] of combinations of failures for the purpose of probability analysis”. DD’s may also be referred to as Reliability Block Diagrams (RBD). The DD shows the same logic as a fault tree using either serial or parallel arrangements of boxes (faults), showing the different paths that could lead to a top event (failure condition). The fault event links represent AND events when organized in parallel and OR events when organized in series. The setup and mathematical formulation are demonstrated using a very simple example shown in Figure 2-8.

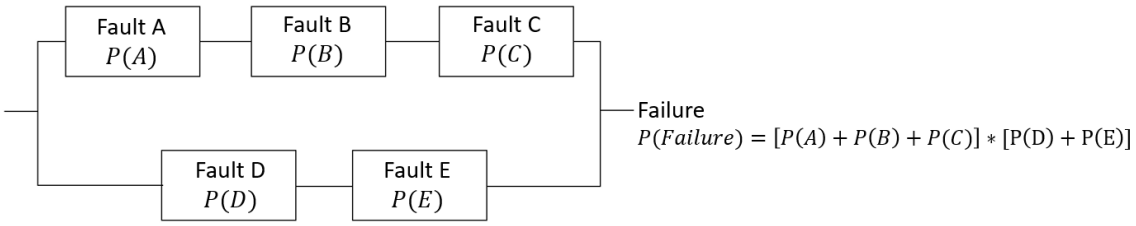


Figure 2-8: Dependence Diagram example

DDs may become very complex, and it may be possible for multiple instances of the same fault to appear in different paths within the diagram (SAE 1996). These represent the common cause failures. Like in FTA, Boolean Algebra and Boolean reduction may be required to ensure probabilities are correctly combined (SAE 1996). A variety of different box types may be used to illustrate different types of failure events. DDs (or RBDs) and fault trees achieve the same goal, and they require the same inputs and knowledge of the system. DDs are particularly useful for showing redundancy, which may not be as obviously visible in a fault tree. They are also subject to the same limitations as described in the discussions regarding FTA.

2.2.5.2 Bayesian Networks (BN)

Bayesian Networks (BN) are becoming more widely used in risk analysis across a variety of industries. They are probabilistic, graphical models of the dependencies between different variables within a system (Villa and Cozzani 2016). The variables of the system are represented using nodes, and the dependence between them is represented using arrows. Each node or variable can be represented by a number of states – these can include failed/working, true/false, or various literal descriptors or numerical values (Smith 2006). Probabilistic calculations can proceed based on the diagram using Bayes theorem of relationships among conditional probabilities, which states that:

$$P(cause|effect) = \frac{P(effect|cause) * P(cause)}{P(cause)}$$

Figure 2-9 contains a simple example of a BN, with the corresponding probability calculations shown (Hartford and Baecher 2004). As the number of variables (nodes)

increases in a BN, the calculation of the probability of the final state becomes increasingly complex.

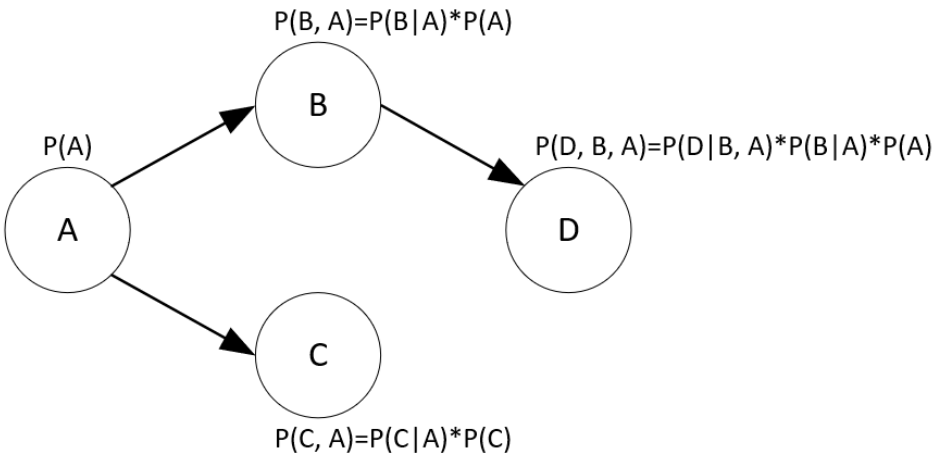


Figure 2-9: Simple Bayesian network example

BNs are capable of dealing with multi-state variables and conditional dependencies which gives them an advantage over other chain-of-event style models like FTA and ETA (Villa and Cozzani 2016; El-Awady 2019). Villa and Cozzani (2016) notes that ETA and FTA can be used as a starting point for development of a BN, and presents a software capable of performing the transformation.

Zhang et al. (2011) applied BN to determine probability of overtopping or internal erosion failures of embankment dams using data available from an embankment dam distress database. The networks developed had a number of different components of the dam and their potential states were either yes/no or a literal descriptor such as satisfactory or unsatisfactory. The goal of the analysis was to determine the probability of failure and sensitivity analyses were performed to determine the most important factors that contributed to the failure modes in consideration (Zhang et al. 2011). Smith (2006) developed a BN for a dam, considering internal erosion and overtopping failure modes. The model developed included variables for precipitation and reservoir level, though it is unclear how these were modelled and whether the approach developed is capable of determining the reservoir level with respect to time. El-Awady (2019) used simulation supported BN to improve the ability of the BN approach to model feedback behaviour. The

approach presented is able to model sub-BN's of particular components in more detail than previous applications. Contributors to failure were identified for case studies within the nuclear and hydropower industries. The approach is promising for determining system vulnerabilities and estimates of probability given limited information. Despite the advances offered by the recent applications of BN, the approach may not be well suited to determine the reservoir elevation (and the values of other variables) with respect to time.

2.2.5.3 Markov Analysis (MA)

Markov Analysis (MA) is another widely-used failure-based method for assessment of system safety and probability of failure. Markov models (also known as Markov chains) are useful for representing the different system states and the relationships between them over time. In Markov models, the transitions between different states are represented by the rates of failure of the different components. The key property of Markov models is that future system states depend only on the current system state, regardless of what led to the current state (SAE 1996). Markov models can be used to represent series systems, parallel systems, and systems which are able to recover and repair themselves.

MA is able to handle common cause failures and interactions in ways that are more challenging using FTA and DD's. They are also able to handle a wider range of system behaviours (SAE 1996). The size of the Markov model grows exponentially in relation to the number of components, which can make MA an extremely complex process.

2.3 Systems approach to safety

The research of Leveson (2011) and Thomas (2012) provide an excellent overview of the limitations associated with the risk-based approaches described in the previous sections. One of the key issues mentioned is that the commonly used techniques focus on failures, which means an entire sub-set of potentially unsafe scenarios may be missing in the analysis. Additionally, the authors state that these traditional risk analysis techniques are

unable to effectively deal with software issues as well as human error and judgement. Commonly used risk analysis tools are often static and/or linear, and as such the ability to determine potentially unsafe scenarios arising from interaction and feedback is limited. Analysis of common cause failures also presents some challenges. Based on these limitations, a systems approach to safety engineering has emerged within the aerospace industry (Leveson 1995, 2011; Leveson et al. 2003), and it is beginning to be recognized and applied in other industries, including nuclear (Song 2012; Thomas 2012), automotive (Vernacchia 2018), railway (France et al. 2018b), software (Pope and Breneman 2018) and dams (Dusil and To 2016; To et al. 2018).

Leveson (2011) utilizes control systems theory to assess several accidents. Many of the examples deal with aerospace and aviation, however examples from other high-profile accidents such as the Walkerton drinking water incident and the Titanic disaster are also provided. Analysis of these accidents led to the development of two generic tools that use a control systems approach to the analyze system safety. The first is Systems Theoretic Accident Model and Processes (STAMP), which was developed for post-accident assessments. The second, Systems Theoretic Process Analysis (STPA), stems from the STAMP technique and was developed for analysis of existing control systems or systems in the design phase. STPA is a systematic process for brainstorming potential control flaws of control systems.

Prior to initiating STPA, the hierarchical control system structure for the system of interest should be developed and the hazards for the system should be defined. This is often done using a flow chart, showing the interactions among elements at different levels of the system. Four general categories of unsafe control actions are provided (Leveson 2011; Thomas 2013):

(1) A required control action not provided or not followed

(2) Unsafe control action is provided that leads to a hazard

(3) A required control action is provided either too late, too early or out of sequence

(4) A required control action is not applied for the wrong amount of time (either applied too long or stopped too soon).

The first step of STPA is to define unsafe control actions for each of the controls in the system of interest. The control actions can be documented using a table such as the example shown in Table 2-7. For each control action identified, the analysts will describe how the situation would unfold and what hazard it could lead to. There may be multiple descriptions under each column for each control action, pertaining to different situations that could lead to a particular unsafe control action being applied and the resultant effects it would have. The second step is to take each of the identified unsafe control actions and identify its causal factors. This is done by using the hierarchical system structure as a guide and looking at the control loop. Investigating the control loop with respect to each unsafe control action can help identify how an unsafe control action might occur – for example, due to incorrect information, a faulty process model or a failed component.

Table 2-7: STPA example table documenting potentially hazardous control actions

Control action	(1) Not provided	(2) Not followed	(3) Initiated at the wrong time	(4) Applied for the wrong amount of time
Control action 1	…	…	…	…

The result of the STPA analysis is a detailed list of what might cause hazards within the system and why. Unlike failure-based methods used in traditional risk assessment, STPA is able to identify non-failure causes of hazards for a system, which makes it a very promising tool for system safety assessment. The approach also does not attempt to estimate probabilities of different outcomes, instead aiming to identify them so they can be addressed or eliminated.

Thomas (2012) advocates the use of Leveson's (2011) tools and presents a methodology for automating the identification of hazards using Leveson's (2011) Systems Theoretic Process Analysis (STPA) model, mainly focusing on potential applications in the nuclear industry. Song (2012) applied the STPA procedure to a specific process at the Darlington Nuclear facility in Ontario, Canada, finding that this procedure enhanced the ability to

identify potentially hazardous conditions at the system level. BC Hydro has recently applied STPA for analysis of dam systems in two applications, the results of which are described by Dusil and To (2016) and To and Dusil (2018). The researchers found that the STPA approach is able to identify vulnerabilities which may be overlooked using conventional risk assessment techniques used for dam safety. It was noted that STPA may not be able to replace existing techniques, but it does provide useful and complimentary insights (Dusil and To 2016).

There are two key limitations of STPA as it pertains to the analysis of dam systems. The first is the natural variability within which the infrastructure is operating and may be attempting to control (inflows, earthquakes, debris build up, ice, etc.). It is not possible to use STPA to determine at what inflow a potentially unsafe situation would become an accident or determine reservoir level response to a set of inflows and operating conditions. The second limitation is that dam systems have components which control the system in a passive way, and STPA was designed for analysis of actively controlled systems. In a dam system, the dams are passively retaining water and the free overflow spillway passively conveys water. Identifying issues that could arise with these passive system components is not possible using STPA. Despite these limitations, STPA does offer a promising tool for addressing issues relating to software and human factors as well as non-failure related causes of potential hazards.

Another systems technique that is becoming more widely used in dam safety applications is simulation. Simulation is a “what if” assessment approach that describes how the system responds to different inputs (Simonovic 2009). A simulation model describes the relationships and interactions between different components within a system, and it can be as detailed as is necessary to achieve its desired purpose. Simulation models contain numerical representations of physical and nonphysical relationships within the system, and may have some type of operating rules in place to determine how controls are applied (Simonovic 2009). Simulation results include a set of outputs, which are the values of the different variables of the system over time. Analysts can experiment with various inputs to determine how the outputs change. The two most common simulation techniques are deterministic and stochastic. In deterministic simulation, a specific set of inputs generates

a specific set of outputs, and multiple runs of the model will always produce the same results. In stochastic simulation, inputs or internal processes of the model may be randomly generated using Monte Carlo techniques. This means that two simulation runs with the same input parameters will produce different results.

Simulation is particularly suited to the problem of dam safety (Hartford et al. 2016). It allows for interactive and dynamic behaviour to be modelled, which is important when considering the different types of both physical and non-physical components in dam systems. Simulation is capable of determining how the system state changes over time (Simonovic 2009), and as such it is the only tool described in this literature review that is capable of directly calculating the reservoir level response to various operating scenarios. Dam system behaviour is highly dependent on the inflows, the initial system state, the states of operating equipment and many other factors – experimenting with these factors through simulation is perhaps the most straightforward way to determine the system response. Simulation allows for an investigation into the emergent behaviour of systems, which results from complex interactions between components and events, and may be difficult to envision by analysing components or sub-systems individually. By modelling the whole system at a sufficient level of detail, the feedbacks and relationships that may lead to emergent behaviour can be incorporated into the model structure.

The potential benefits of simulation and the systems approach in general are becoming recognized within the dams industry. Regan (2010), Baecher (2013), Komey et al., (2015), Micovic et al. (2015), and Hartford et al. (2016) all advocate for the consideration of dams as systems. Baecher et al. (2013) present a stochastic simulation methodology framework for dam safety flow control analysis. Hartford et al. (2016) present two examples that utilize a systems approach embedded within a stochastic simulation to determine the likelihood of failures for dam systems. One of the examples described by Hartford et al. (2016) and developed by Komey (2014; 2015) involves stochastic simulation of hydropower dam response to disturbances such as ice, debris, and human intervention on the Mattagami River System in Ontario, Canada. The approach utilizes the GoldSim Monte-Carlo modelling platform to determine various impacts these disturbances may have on safe operation of the system (Komey 2014; Hartford et al. 2016). A probabilistic

framework is used to model disturbances such as ice and debris, with fragility curves and simple failure rates defined to determine the probability of gate or turbine failure, and gamma distributions to determine time to repair (Komey 2014; Hartford et al. 2016). (Zielinski et al. 2016) use a similar approach to Komey (2014; 2015) to assess the safety of the Madawaska River System in Ontario, Canada, using a 10,000 year continuous simulation to estimate the probability of failure for each dam in the system. Another example described by Hartford et al. (2016) involves a system dynamics model of the Göta River System in Sweden, with the system built up in layers of increasing complexity. The model can be run in either stochastic or deterministic mode and is used to investigate system response to sea level fluctuations, landslides and climate change.

These probability-driven stochastic model examples help address many of the shortcomings of traditional risk assessment approaches. Dynamic, nonlinear behaviour can be captured by these models and they can be developed to be as complex as necessary to more realistically represent the system of interest. One limitation of the stochastic simulation approach is that it requires a very large number of simulation years in order to assess combinations of component operating states that have a very low probability of occurring together. There is no way of assuring the modeller that a complete set of possible operating states has been captured in the simulation. The operating state combinations that arise from a stochastic simulation model will differ between two different runs of a simulation with the same inputs. Beyond some certain limit, if the stochastic simulation is run for long enough, there would be a complete set of possible operating scenarios. However, there would be a significant amount of time and resources spent simulating conditions where nothing is wrong with the system. Given the large number of potential combinations of operating states and current computational abilities, a full assessment of all scenarios using stochastic techniques is not currently possible. Despite these limitations, the work of Zielinski et al. (2016) and Komey (2014, 2015) provide a good indication that a shift in focus is required from extreme events to events occurring within the design envelope that might actually contribute more to the overall system risk.

2.4 Summary

In the preceding sections of this chapter, a number of tools were presented that are commonly used within the dams industry to analyze system risk. A description of the advantages and limitations of each approach was provided. Ultimately, the more commonly applied techniques in risk analysis have served the dams community well. Hartford et al. (2016) suggest that the risk based approach has significantly improved the understanding of dam safety in a number of ways. These include facilitating analysis of less easily analyzed failure modes such as internal erosion, highlighting the importance of analyzing human factors, and indicating that the extreme events required for dam design may not be the most significant contributors to risk. There are, however, limitations to the most commonly applied techniques. It would be extremely challenging using these approaches to analyze the combinatorically large number of possible events that may possible occur. Interactions and feedbacks are typically simplified or omitted using the traditional techniques, meaning dynamic behaviour cannot be effectively analyzed. Many issues arise when dealing with human factors, software errors and design flaws. Analysis of time-considerations is also beyond the scope of applicability of these existing approaches. Finally, many existing approaches omit further consideration of certain combinations of events which have a low combined probability – despite there being enough of these combinations to add up to a significant risk to the system. The following paragraphs detail the main conclusions from the assessment of current practices in risk analysis.

FMEA is a tool for determining how components of a system can fail and what their causes and effects will be. The effects of failures at one level of the system can be determined on the next level up until the entire system is analyzed. This approach is a useful tool for brainstorming and determining potential disturbances which create the constraints within which the system may operate. However, FMEA is a failure-based method that may miss a sub-set of potentially unsafe scenarios that are not triggered by failures. It is not able to systematically determine combinations of constraints that could be encountered in system operation. It is also unable to determine and quantify the reservoir level response and has presented some challenges when dealing with complexity, feedback and interaction within

hierarchical systems (the IEC standard on FMEA states that limiting analysis to a maximum of two levels of hierarchy is good practice).

PFMA is a useful tool for looking at systems as a whole and brainstorming potential failure modes which could develop at the system level. However, it is a completely heuristic approach, and does not explicitly involve analysis of various levels of a hierarchical system or the interactions between the levels. It does not facilitate quantification of system behaviour and may miss certain components which are considered to be of less importance to the analysts due to perceived low consequence or likelihood. It has the same limitations as FMEA and relies more heavily on expert judgement and subjectivity.

ETA and FTA are both very practical tools for quantitative probabilistic assessment of failures and their impacts. However, these approaches are failure-based, linearize the progression of events and are unable to easily deal with feedback and nonlinear interactions. ETA and FTA also begin with initiating events, and top events (faults), respectively, which must be predetermined in some way. There is a very serious challenge using these approaches in analyzing combinations of events, of which there may be an extremely large number of possibilities. This is not a challenge that will be easily overcome given the current state of the science. Finally, these approaches are not able to determine the reservoir level response to various operating conditions due to their inability to analyze component interactions and feedback behaviour.

Ultimately, the existing approaches, while useful, may not be adequate to capture the dynamic behaviour of complex, interacting hierarchical systems. Because of the recognized limitations, a systems approach has begun to emerge, and is beginning to gain some momentum within the dams industry for analysis of system safety (Hartford et al. 2016). STPA is an excellent tool for analyzing potential control flaws in complex, actively controlled systems. The key limitations of STPA that pertain to the analysis of dam systems are that (a) dam systems may have many safety-critical components that provide passive control, and (b) dam systems are acting to control natural inflows, so determining the dynamic system response to the inflows is necessary to get a complete picture of system safety. STPA is unable to determine the reservoir level response to various conditions.

Nevertheless, STPA does provide a good starting point for the assessment of the actively controlled components of a dam system. It provides very useful information for determining potential system operating constraints (scenarios) for actively controlled components in a systematic way and includes both failures and non-failures.

Simulation is another tool that can be used in the systems approach, and it is becoming more commonly applied within the dams industry for safety analysis. Hartford et al. (2016) focuses on several applications of stochastic simulation. In stochastic simulation, random failures of components may be initiated (with random outage lengths) to determine the overall probability of failure for the system. In this way, each run of a stochastic simulation model would produce a different output. If run for enough years, the probability of failure would begin to converge on a single value. The approach is becoming more widely applied for dam safety analysis (Komey 2014; Komey et al. 2015; Hartford et al. 2016). Stochastic simulation addresses more of the research requirements described previously than other safety assessment approaches for dam systems. Stochastic simulation can capture dynamic feedback relationships between system components if the system is modelled in adequate detail. Simulation outputs for a dam system can include the reservoir level fluctuations in response to various inflows and constraints, which makes simulation a particularly promising tool for dam safety analysis. It is also possible to assess potential "combinations of events" using stochastic simulation, though the ability to do so is limited by the length of the run (computing power). Because the probabilities applied to the events (equipment states) are relatively low, multiple events occurring and impacting one another are very rare within a stochastic simulation if not run for enough years. In theory, stochastic simulations run for enough simulation-years would eventually cover all of the possibilities, however the computational requirements to achieve a complete coverage of the possibilities would be beyond current capabilities. As such, the existing implementations of stochastic simulation may not be able to capture a complete set of possible combinations of component operating states at the current time. Stochastic simulation has the benefit of easily estimating the overall probability of flow control failure of a system, though the assessment of criticality for specific scenarios would require the use of data mining techniques as well as extremely large number of simulation-years. Some of the current

limitations associated with a purely stochastic approach can be demonstrated using a simple example.

Consider a system with five components, A, B, C, D and E. Assuming each component is either functioning or failed, there are 2^5=32 potential combinations of failures as follows (normal component states are not shown):

No failures, A, B, C, D, E, AB, AC, AD, AE, BC, BD, BE, CD, CE, DE, ABC, ABD, ABE, ACD, ACE, ADE, BCD, BCE, BDE, CDE, ABCD, ABCE, ABDE, ACDE, BCDE, ABCDE.

For this example, assume that the goal of the stochastic simulation is to generate all possible combinations of the component operating states at least once. Assuming that the probability of failure for each component is 0.1% per day, and the model is run for as many years as necessary at a daily time step until each combination has been simulated at least once, the number of years required to arrive at each combination is shown in Table 2-8, which also shows the corresponding number of years within which each combination was simulated.

Obviously, the combinations (scenarios) with less failures have a higher probability and are simulated more frequently than the combinations with a higher number of failures. It is also worth noting the high number of non-failure years simulated. Averaging over 50 total runs, the simulation spends about 25% of its time simulating non-failure years. This number is dependent on the assumed failure rates and will increase as the assumed failure rates decrease. The amount of effort spent simulating each scenario is a function of its probability, so some of the worse scenarios are focused on less because of their low likelihood. About half of the possible scenarios (the lower, less probable part of the list) are simulated less than 40 times, which means the simulation focuses less than ~0.5% of the simulation effort on those scenarios. A very large number of simulation-years would be required to collect enough data with which to assess the criticality of these more severe and less likely combinations of operating states.

A volumetric representation of the system's "possibility space" is another useful way of demonstrating how stochastic simulation samples different events. The "possibility space"

Table 2-8: Stochastic simulation of operating state combinations

	Run									
Combination	1	2	3	4	5	6	7	8	9	10
A	1007	485	319	685	1152	667	328	310	498	202
B	922	479	316	674	1132	624	341	313	517	176
C	952	475	312	649	1191	684	349	289	510	191
D	955	462	293	701	1149	625	324	299	508	180
E	984	462	266	683	1183	668	362	295	518	198
AB	192	91	58	125	219	130	57	58	87	40
AC	205	95	55	130	192	138	86	68	104	35
AD	183	77	43	131	212	144	60	49	103	32
AE	172	97	56	128	263	107	72	61	98	40
BC	195	111	61	142	233	118	69	56	105	30
BD	185	95	56	143	225	124	59	59	88	38
BE	184	93	67	126	221	139	68	55	103	32
CD	190	84	65	129	252	145	57	60	112	50
CE	198	92	68	135	218	133	66	57	110	43
DE	188	87	54	135	233	139	77	51	102	42
ABC	36	24	9	29	40	25	11	9	24	7
ABD	46	24	7	27	44	26	18	12	22	8
ABE	34	20	12	24	54	22	11	20	15	8
ACD	45	19	9	25	43	26	14	8	26	6
ACE	31	21	12	36	50	21	13	12	20	6
ADE	30	18	7	24	39	28	15	8	16	9
BCD	44	14	19	29	36	25	11	14	22	6
BCE	34	13	12	29	49	23	14	10	20	9
BDE	25	20	9	16	51	29	12	14	25	8
CDE	39	13	9	15	59	26	14	15	31	7
ABCD	7	3	2	2	6	2	2	1	2	2
ABCE	4	7	3	3	5	6	3	3	3	2
ABDE	10	6	6	7	9	7	1	2	7	1
ACDE	8	2	3	6	3	10	5	5	4	2
BCDE	5	2	1	6	14	5	1	1	5	2
ABCDE	1	1	1	1	1	1	1	1	1	1
Total number of failures simulated	7111	3492	2210	4995	8578	4867	2521	2215	3806	1413
Total number of non-failures simulated	4814	2428	1483	3433	5801	3206	1719	1597	2506	968
Total	11925	5920	3693	8428	14379	8073	4240	3812	6312	2381

is a visual representation of the full realm of physically possible system states and their frequency of occurrence. In Figure 2-10 (King and Simonovic, 2020), the frequency, F_N of N components in an adverse operating state (N_U) is plotted against the frequency, F_D of outage durations D_U. Simple FN relationships are assumed and a 3-dimensional possibility space is created. The planes of adverse component operating states are represented using red, orange, green, blue and purple outlined areas. It is important to note that this possibility space is a very simplified representation of the problem – the possibility space would also include inflows, starting reservoir elevations and timing of component outages. Nevertheless, using this simplified example figure, the stochastic samples can be plotted (shown as black dots). Each dot represents one sample that could be stochastically generated. The dots are centred around zero component outages, which have a higher frequency, zero components out of service and zero outage length. While the samples do extend into the outer reaches of the possibility space, they provide the best coverage of the higher-frequency scenarios (zero to one components unavailable). The coverage of the less probable, more extreme scenarios (where more components are out of service) is limited by the number of years for which the simulation is run. In a stochastic simulation, this volume is predefined, since the components, their outage frequency and their outage length frequency are inputs to the model.

In conclusion, neither the more commonly used risk assessment approaches or the existing techniques utilized from the systems approach are currently capable of systematic and dynamic evaluation of the combinations of events that can lead to flow control failure in dam systems. A new methodology is required, building on the existing tools from the systems approach to define, analyze and evaluate a more complete range of potential combinations of events (scenarios). The approach must be able to handle complexity, feedback and nonlinear behaviour. Dynamic indicators of the system performance are a required output of the analysis, as well as parameters that can be used to rank the relative importance of a large number of potential scenarios.

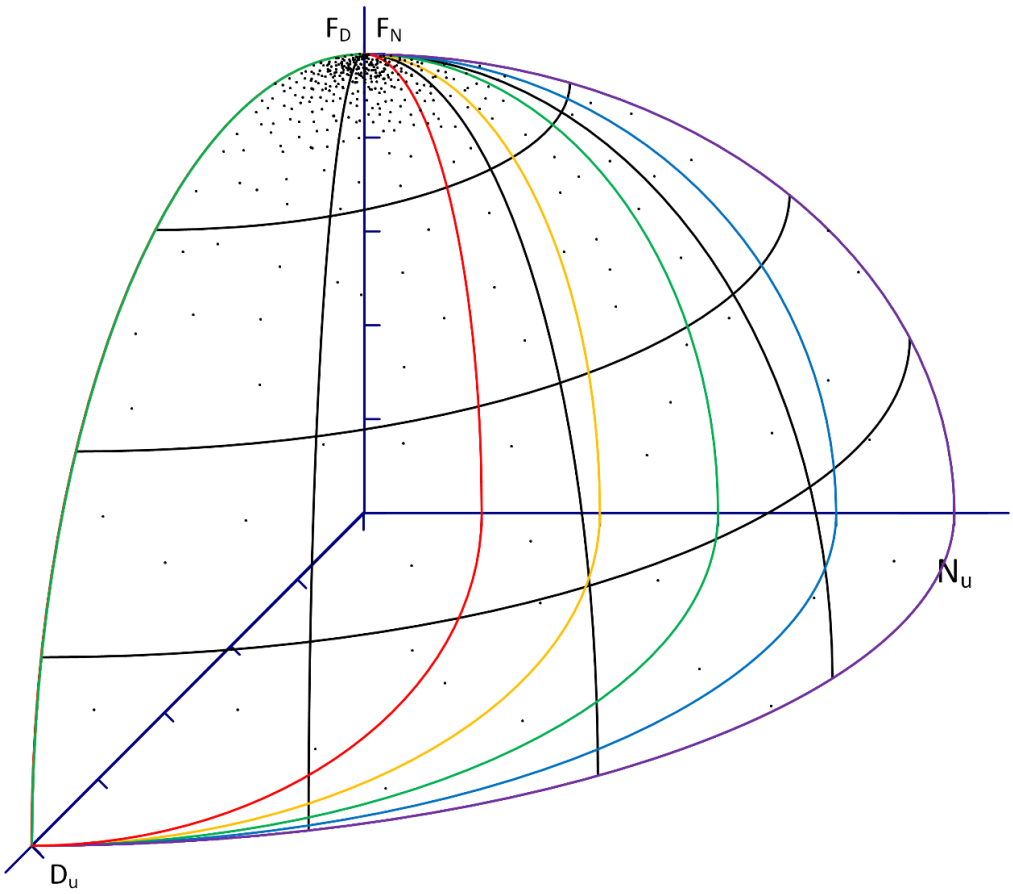


Figure 2-10: Stochastic sampling from within the possibility space

Chapter 3

3 Methodology

In this chapter, an improved methodology for assessment of flow control in dam safety is developed. The methodology draws on the benefits of existing approaches where possible, making improvements that can facilitate a more thorough understanding of system response to a more complete set of scenarios. The next section describes the methodology justification and development, followed by a complete description of the methodology steps.

3.1 Justification and development

A dam system is fundamentally an open control system. It is forced by inputs (inflows) which vary with time in a relatively predictable way. Dam system outputs (outflows, energy, etc.) are also constrained in relatively predictable ways, but random deviations within the system may occur that affect the system outputs. For example, a spillway gate can open or close to release the desired amount of flow downstream, but failure of infrastructure which supports the gate function can cause the output constraints to deviate from their normal values. There is a need within the dams industry to better understand how dam system outputs may be constrained and what possible system outcomes may result. Specifically, determining the reservoir level and outflow response to the full range of possible operating constraints is an important goal that can help dam owners better understand vulnerabilities within the system and determine appropriate courses of action to address them. Using the control system structure presented by Leveson (2011) and modified for a dam system, a basic mathematical framework can be developed for calculation of the reservoir storage over time (which is directly related to the elevation through a stage-storage curve). This is shown in Figure 3-1. The boundaries of the system are the point at which the inflows enter the reservoir and the point at which the outflows leave the dam. Included in the system is all of the infrastructure at the dam site, including dams, gates, gate actuators, sensors as well as information relay, processing, operational decision making and implementation of operations (which may take place off site).

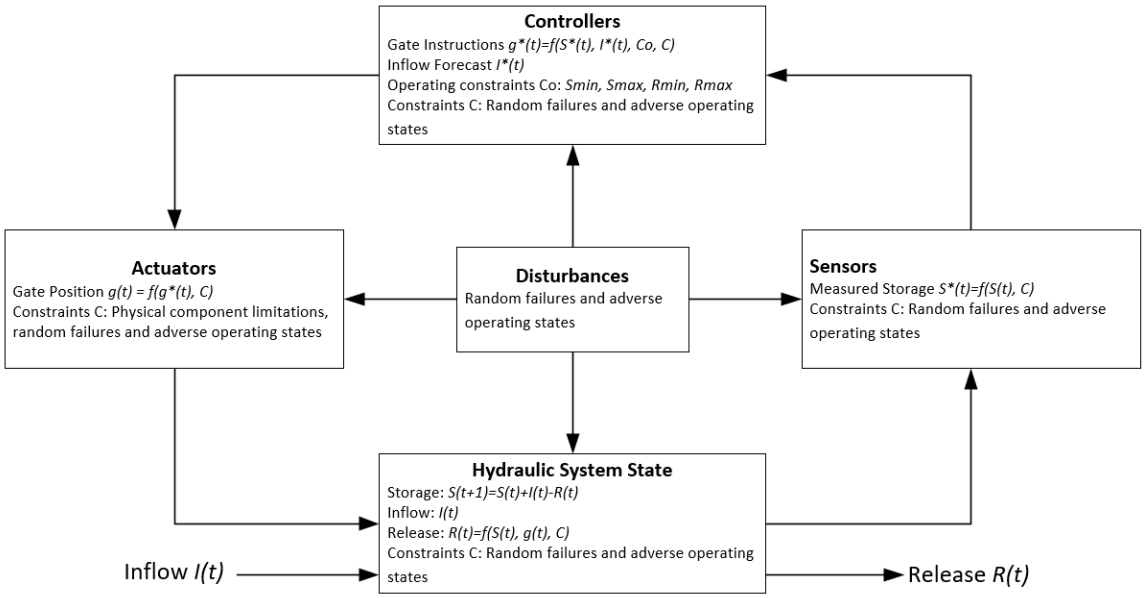


Figure 3-1: Mathematical framework for determining dam system behaviour

The control loop shows how information is passed through the system and what the main connections are between different sectors (feedbacks). The relationships between Storage, Inflow and Release are easily represented by basic mathematical equations and rule curves for decision making with respect to controlled flow releases. This type of modelling is done frequently for operations planning and analysis of dam systems. The area where more work is needed is in determining constraints that come into play in several sectors of the system and may impact measured system state values, operational decisions, operability of equipment and capacity of water passages. The goal of this research is (a) systematically defining these operating conditions and (b) understanding how they may adversely impact the system state and outputs. A new approach is necessary with the following goals:

- Reduced reliance on subjectivity and expert judgement.
- Ability to determine potential constraints on system operation.
- Ability to determine potential constraints that *are not limited to* failure modes.
- Automatic generation of potential combinations of constraints.
- Determine the likelihood of constraints without significant simplifying assumptions.
- Quantification of the dynamic system response to constraints. Specifically, how the reservoir level and outflows change with time for a given set of constraints (component operating states), constraint parameters and inflows.

- Inclusion of feedbacks, interactions and nonlinear behaviour.
- Ability to deal with system complexity without the use of extreme simplifications that undermine the results.
- Ability to deal with uncertainty in the outcomes.
- Estimation of criticality for a given scenario.
- Estimation of overall probability of flow control failure for the system.

Table 3-1 contains a list of the main approaches discussed in the literature review, evaluating them within the context of these research goals based on the observations in the previous chapter. It is important to note that many of these tools may not be utilized independently. The results of FMEA, for example, can be used as inputs to FTA or ETA. Combining the tools may result in improved ability to achieve these research goals, however the key limitations remain for the majority: an inability to model dynamic system behaviour, complexity, feedback and interaction. The most promising tool in terms of the aforementioned research requirements is the stochastic simulation approach, described in detail by Hartford et al. (2016). The key limitation of the stochastic simulation is that it spends a significant amount of effort simulating non-failures, and as such may not be an efficient means of systematic evaluation of each possible combination of events. The methodology proposed in this work seeks to determine (a) what the possible combinations of events are, (b) what their range of impacts might be and (c) what their relative importance (criticality) is, with respect to other scenarios.

Systematic analysis of each possible scenario, in theory, may be achieved using a completely deterministic simulation of predefined scenarios. However, the timing of the scenario's predetermined adverse operating states (events) and determining whether events influence one another significantly complicates the analysis. To completely and deterministically analyze the full range of possible outcomes of a single scenario, all possible combinations of event timing and inflows should be considered. Consider an example scenario with three events, A, B, and C, and 10,000 years of possible daily inflow values (this number of inflow-years is selected, in theory, to include inflows up to the PMF which has an annual exceedance frequency of 1 in 10,000 years). There are a total of $365 * 10{,}000 = 3{,}650{,}000$ possible inflow start days. Assuming the events can happen any time

Table 3-1: Overview of approaches and their applicability to the research problem

Requirement	FMEA	PFMA	ETA	FTA	STPA	Stochastic Simulation
Reduced subjectivity	Slightly (systematic process)	No (fully heuristic)	No	No	Slightly (systematic process)	Slightly (simulation model automatically determine system outcomes)
Determine constraints on system operation	Yes	Yes	No – Requires this upfront	No – Requires this upfront	Yes	No – Requires this upfront
Ability to address non-failure related constraints	No – Failure based method	No – Failure based method	No – Failure based method	No – Failure based method	Yes	Yes – If non-failures included in potential constraints
Automatically determine potential combinations of constraints (scenarios)	No	No	No	No	Yes	No
Determine likelihood of constraints without significant simplifying assumptions	No	No	No	No	No	No
Quantification of dynamic system response	No – Static analysis	No – Static analysis	No – Linear chain-of-events	No – Linear chain-of-events	No – Static analysis	Yes – Dynamic analysis
Inclusion of feedbacks and nonlinear behaviour	No – Static analysis	No – Static analysis	No – Linear chain-of-events	No – Linear chain-of-events	Includes feedbacks but does not analyze system behaviour	Yes – Dynamic analysis that can include feedbacks and nonlinear behaviour
Ability to handle complexity	Somewhat, can pose challenges	Somewhat, can pose challenges	Somewhat, can pose challenges	Somewhat, can pose challenges	Yes	Yes
Assessment of uncertainty in the outcomes of a scenario	No – Does not analyze scenario outcomes	No – Does not analyze scenario outcomes	Yes – Can experiment with assumptions	Yes – Can experiment with assumptions	No – Does not analyze scenario outcomes	Not directly – Could potentially assess specific scenarios using data mining
Ability to deal with common cause failures	Limited to qualitative only	Limited to qualitative only	Limited – requires careful consideration	Limited – requires careful consideration	Yes	Yes
Estimation of scenario's criticality	No – Static analysis	No – Static analysis	Yes	Yes	No – STPA is generally qualitative	Not directly – Could potentially assess specific scenarios using data mining
Estimation of overall system flow control failure probability	No – Static analysis	No – Static analysis	Yes, if all possible combinations included	Yes, if all possible combinations included	No – STPA does not involve probabilistic assessment	Yes

within a one-year window, there are a total of $365 * 365 * 365 = 48{,}627{,}125$ possible combinations of event initiation times (the day in which the adverse operating state begins). This means $3{,}650{,}000 * 48{,}627{,}125 = 1.77 \times 10^{14}$ possible ways to execute the simulation for a single scenario with three events occurring. This number considers only one set of possible impacts for event A, B, and C, which may have impacts (for example, outage lengths), that can significantly vary. Clearly, the number of combinations and scenarios that must be analyzed to fully define the suite of potential outcomes for a system becomes computationally prohibitive, even with state-of-the-art computing technology such as cluster computing. Monte Carlo selection of event timing and inflow start day from s synthetically generated inflow record can be useful to sample a small number of these possibilities and dynamically characterize some of the possible outcomes for a given scenario. Each predetermined scenario can be simulated through a number of Monte Carlo iterations to better understand the possible range of outcomes resulting from that scenario. This is the hybrid Deterministic Monte Carlo approach proposed in this research, where events (component operating states) are pre-selected and their impacts, timing and inflows are varied to better understand the possibilities within current computational capabilities. The scenario inputs represent the deterministic portion of the model, and the varying of scenario parameters represents the Monte Carlo portion of the model.

The approach presented in this thesis aims to provide a more even coverage of the possibility space, as shown in Figure 3-2 (King and Simonovic, 2020). The sample dots are color coded, to indicate which “adverse component operating state plane” the samples are taken from. The Deterministic Monte Carlo approach forces the simulation to take samples from within each plane, because it does not rely on the frequencies of failure and duration to generate the samples. Each plane represents a single scenario, and the scenarios are predetermined and simulated regardless of their likelihood. For the Deterministic Monte Carlo approach, only a single sample would be taken to represent “normal” conditions. In the proposed approach, the frequencies of the component outages and outage lengths are not required to run the model, so the frequency dimension of the possibility space is not defined. If enough information is available to define these frequencies, a complete probabilistic analysis using the results of the analysis is possible.

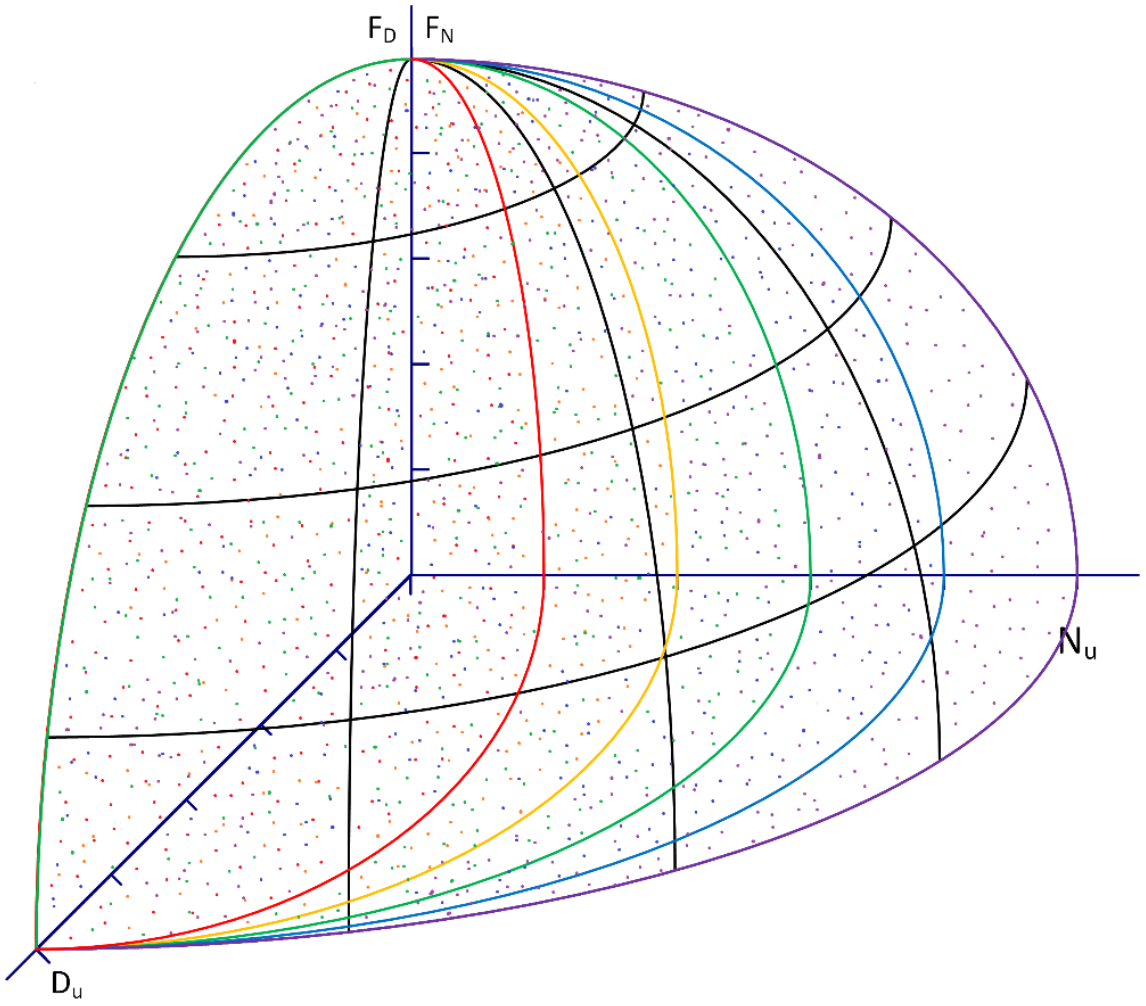


Figure 3-2: Deterministic Monte Carlo sampling from within the possibility space

Another key timing related issue that must be considered is the problem of whether preceding events are influencing the results of subsequent events. Such considerations arise when a component failure has been rectified, but the overall system remains in a "disturbed state", that is, the system has not been restored to the state that it would have been in if the component failure had not occurred. This means that the "system state deviance" must be a factor to be considered along with the timing of component failures. Since "system state deviance" is determined by operational decisions, (eg. The decision to release water to return to a normal state), these decisions must be somehow included as factors in determining the extent of the deviance. This can be achieved by analyzing whether the reservoir level has returned to a predefined "normal" state following the initiation of an

event. If not, there may be independent sub-scenarios within the simulation that should not count towards results of the scenario being analyzed. Consider the example shown in Figure 3-3 (King and Simonovic, 2020), which has three events A, B and C occurring within some time of one another.

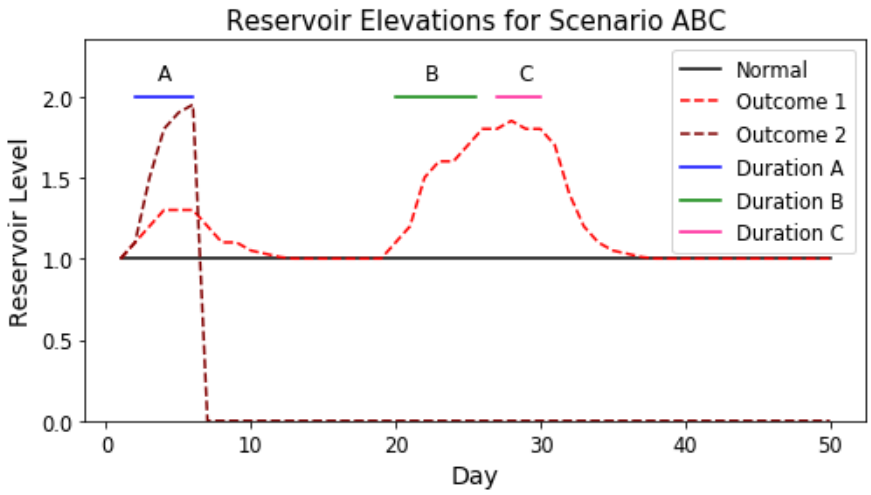


Figure 3-3: Example output reservoir elevations for Scenario ABC (King and Simonovic, 2020)

In the example, the reservoir has a constant elevation of 1 m under normal circumstances where everything is operational. The event outage occurrence dates and lengths (durations) are represented by the horizontal lines above the plots. For Outcome 1 (light red), Event A causes an increase to about 1.3 m and then the reservoir level returns to the normal elevation of 1 m prior to the initiation of Event B. The rate at which it returns to the normal elevation would be determined by the operations planning algorithm within the simulation model. Event B causes in increase in reservoir elevation to about 1.8 m, after which Event C begins and increases the reservoir a further 0.1 m. After Event C, the reservoir returns to its normal elevation as a result of the operations planning decisions. In Outcome 1, Event A does not have any impact on the outcome of Events B and C, because the reservoir level has returned to a normal elevation. Event B, however, does impact Event C. Thus, for Outcome 1, two sub-scenarios are observed: (1) The result of Event A, and (2) the combined result of Events B and C. In Outcome 2 (dark red), the reservoir rises to about 1.95 m following Event A, at which point dam breach is triggered and the reservoir drops

to 0 m in elevation. In this case, the only sub-scenario being analyzed is Event A. This example shows that despite the simulation being intended to analyze the combined impacts of Events A, B and C, they cannot be assumed to be influencing one another. Some analysis of each simulated outcome (the reservoir levels from each simulated Monte-Carlo iteration) is required to ensure simulation results are attributed to the actual scenarios being represented within the analysis. “Complete iterations” are considered to be the subset of scenario results where all of the scenario’s events both occurred and affected one another.

In using the proposed Deterministic Monte Carlo approach, it is important to consider that the goal of the exercise is to analyze all scenarios (predefined combinations of operating states) as completely as possible given computational time constraints, to determine the criticality of these combinations and identify particularly vulnerable components. There should be enough data on each scenario to estimate the range of expected system performance as well as the failure frequencies, failure inflow thresholds and reservoir level exceedance frequencies. To ensure there is enough data collected for each scenario, it may be necessary to limit the time between events to ensure their collective impacts can be assessed. This limit may be determined as a function of the impact lengths for a given iteration (for example, by taking the sum of impact lengths). Whether the event initiation time limit should be more or less than the sum of the impact lengths requires experimenting with scenarios to determine how long the system typically takes to return to normal operation. For “flashy” reservoirs with relatively limited storage compared to inflows, the recovery time following a return to normal operations may be quite short – days or even hours. For reservoirs with large storage in comparison to inflows, this recovery time may be significantly longer. The recovery time may also be less than the sum of impact lengths, due to inflows that are less than the total capacity of the available flow conveyance facilities. The recovery time should influence the modellers decision regarding the appropriate time limit for event initiation.

In summary, the methodology developed in this work uses a systems approach for dam safety analysis, attempting to draw on the strengths of the existing techniques, combining and building on them with the goal of addressing the key element that is missing in all of them – assessing outcomes from a large number of the possible combinations of events.

This research focuses on identification and analysis of a more complete subset of potentially unsafe scenarios than has previously been considered in dam safety analysis. A Deterministic Monte Carlo simulation approach is proposed, in which scenarios are systematically defined upfront and used as a deterministic input to the model. Defining the complete set of failures and events to be simulated ensures all combinations of the defined component operating states (or constraints) are evaluated. It also means that the simulation efforts are divided equally between each scenario (combination of operating states), so a more thorough analysis of each scenario is possible than using a purely stochastic approach. Finally, the proposed approach reduces the amount of time spent simulating non-failures. The scenario parameters, such as the timing and magnitude of impacts, are varied using Monte Carlo techniques so that each scenario is run as many times as possible given computational time constraints. Varying impact parameters allows some analysis of the uncertainty relating to the estimate of scenario impact magnitudes (for example, it is hard to estimate how long a component will be out of service, so a range of different values can be tested). Conditional probabilities of failure and reservoir level exceedances above key elevations (given a scenario has occurred) are direct outputs of the simulation. In this way, a much larger subset of the events that contribute to the probability are analyzed. Focusing on numerically assessing the entire design envelope and the complete range of possibilities can help asset owners in becoming more prepared for any event (or combination of events), regardless of its probability.

A flow chart detailing the overall methodology is shown in Figure 3-4 (King and Simonovic, 2020). First, a component operating states database is created, which defines individual components and their operating states, causal factors and potential range of direct impacts. These represent the constraints within which the system may have to function. Population of the database requires extensive knowledge of the system and is similar to a FMEA but also includes non-failure operating states. A combinatorial procedure uses the database entries to come up with the complete range of potential scenarios for the system, which are used as inputs to a simulation model. Synthetic climate data is generated for the system of interest and used in a hydrological model to develop a long, synthetic timeseries of inflows to be used as inputs in the simulation model. The scenarios become the inputs for a deterministic simulation model which is run many times

for each scenario, with Monte-Carlo generated inputs that vary the inflows and incident timing, as well as the scenario impacts. Outcomes are descriptors of the system behaviour over time, including the releases through various conduits as well as adverse impacts, such as dam overtopping, uncontrolled flow releases, or dam breach. Outcomes for each scenario are assessed and aggregated performance measures for scenario groups are computed. The results of the analysis could be utilized to develop or refine response and mitigation strategies to improve system performance. Further analysis may be possible that would allow for overall estimates of the probability of failure for each individual scenario, and probability of failure for the system as a whole – this would require probabilistic analysis of operating states, which is a complex task that is not explored in this research.

The proposed methodology has been developed to meet as many of the requirements in Table 3-1 as possible within the time frame of the work. The key missing pieces are that (a) it still inherently relies on subjectivity in the population of the operating states database and development of the simulation model – there is currently no substitute for expert knowledge and engineering judgement; (b) the likelihood of operating states are still difficult to estimate without significant simplifying assumptions; and, (c) the approach does not directly result in estimates of overall probability of flow control failure for the system, though with some additional analysis this may be possible. There are also an extremely large number of configurations (with different inflow sequences, event timing, and impact magnitudes) for each combination of operating states, as described earlier in this section – this approach covers only a small subset of the possible configurations for each scenario, through the use of Monte Carlo techniques. This means that results between two identical simulations of a given scenario would vary, and the results are only estimates of the criticality of each scenario, given computational time constraints.

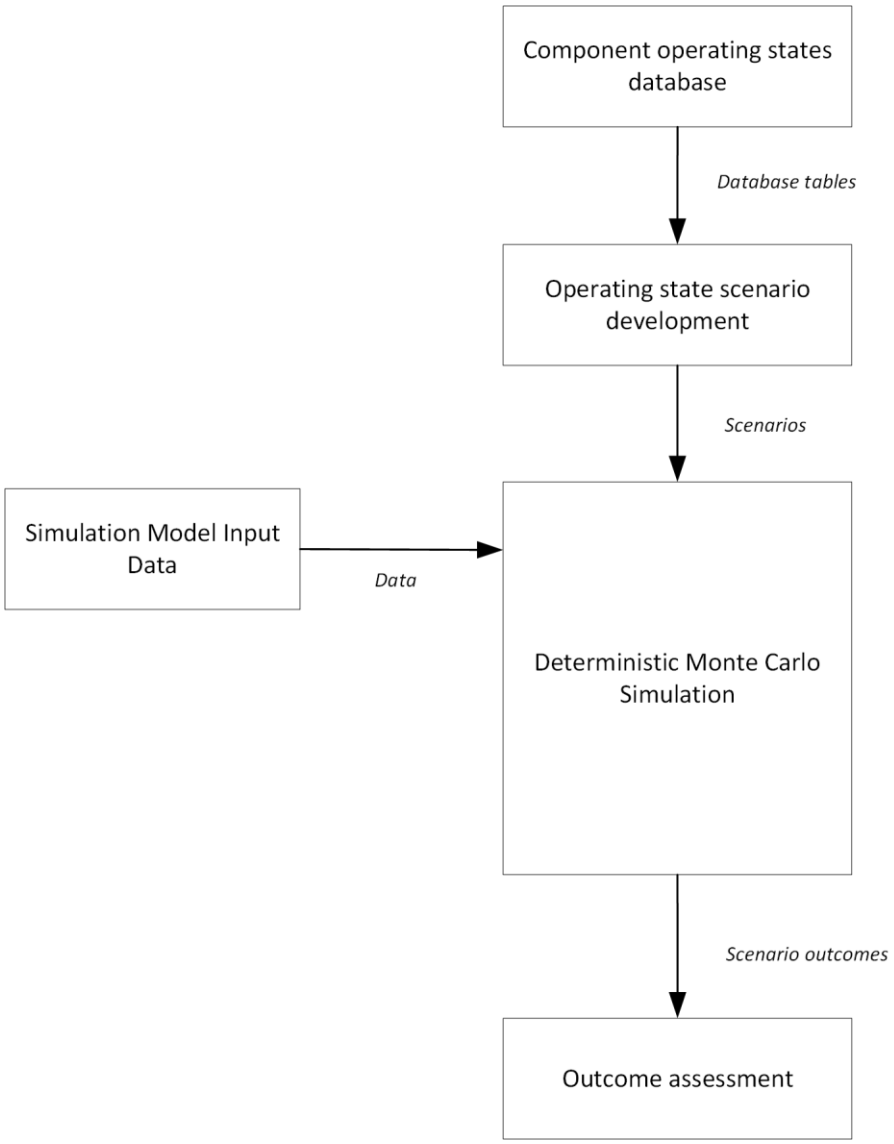


Figure 3-4: Overall methodology flow chart (King and Simonovic, 2020)

Each of the methodology steps are described in detail in the following sections. First, a description of the components operating state database and the process for database population is provided. Next, scenario development is discussed with a description of the combinatorial procedure developed in this research. Then, a detailed description of the Deterministic Monte Carlo simulation approach is presented, which uses a system dynamics simulation model with Monte-Carlo varied inputs. The following section contains a description of the modelling approach used for inflow generation. Finally,

scenario outcome assessment is discussed including the selection of criticality parameters and performance measures.

3.2 Component Operating States Database

A component operating states database is used to define the operating states for each system component and the causal factors that could lead to the development of the operating states. The approach of database population is similar to FMEA but also considers non-failure related operating states, including normal or functional operating states. Population of the database can significantly benefit from the application of (a) the STPA technique for actively controlled components and (b) a single-level FMEA analysis for system components. There is still a significant amount of reliance on expert judgement, but this may be slightly reduced if the systematic approach of STPA is used to inform database population.

In order to develop an exhaustive list of potential operating scenarios for a hydropower system, each individual component of the system (whether it be physical or non-physical) must be analysed and its modes of operation considered. This is achieved using an operating states database, which was designed using a relational database software called Microsoft SQL Server (MSSQL). MSSQL is a software used to generate and populate computer databases. It can be used to generate interfaces that assist with database population and information access. Using MSSQL, data are organized into relational tables to model aspects of reality, such as the system elements, at different levels (system, component and reservoir), the operating states and their characteristics as well as the causal factors. Another important feature used in the design of the database is the *store procedures*, which include functions that provide flexibility for developers, and are used to insert and recover data very efficiently from the database with less computational burden. In addition to this, there are *views* which allow the combination of several tables in a relational way and return aggregated data to the user interface. The structure and the entity relationship (ER) diagram of the database is shown in Figure 3-5 which depicts the relationships between various levels of the system. The lines connecting the objects (tables) in Figure 3-5 represent “foreigner keys” which ensure data consistency and integrity. The database was designed to facilitate simple data entry using a web-based user interface. It

may also be accessed, modified and queried using MSSQL Server Management Studio. The design of the database is meant to be as general as possible, facilitating the analysis of various dam systems with different configurations. Figure 3-5 presents three major groups of tables organized as: a) *System elements*: containing the tables representing the System, Component and Reservoir level elements; b) *Operational States*: containing the tables representing the operating states for each of the system elements; c) *Casual Factors*: containing the tables storing the causal factors; d) *Auxiliary objects*: containing tables that are used to store user accounts and system logs.

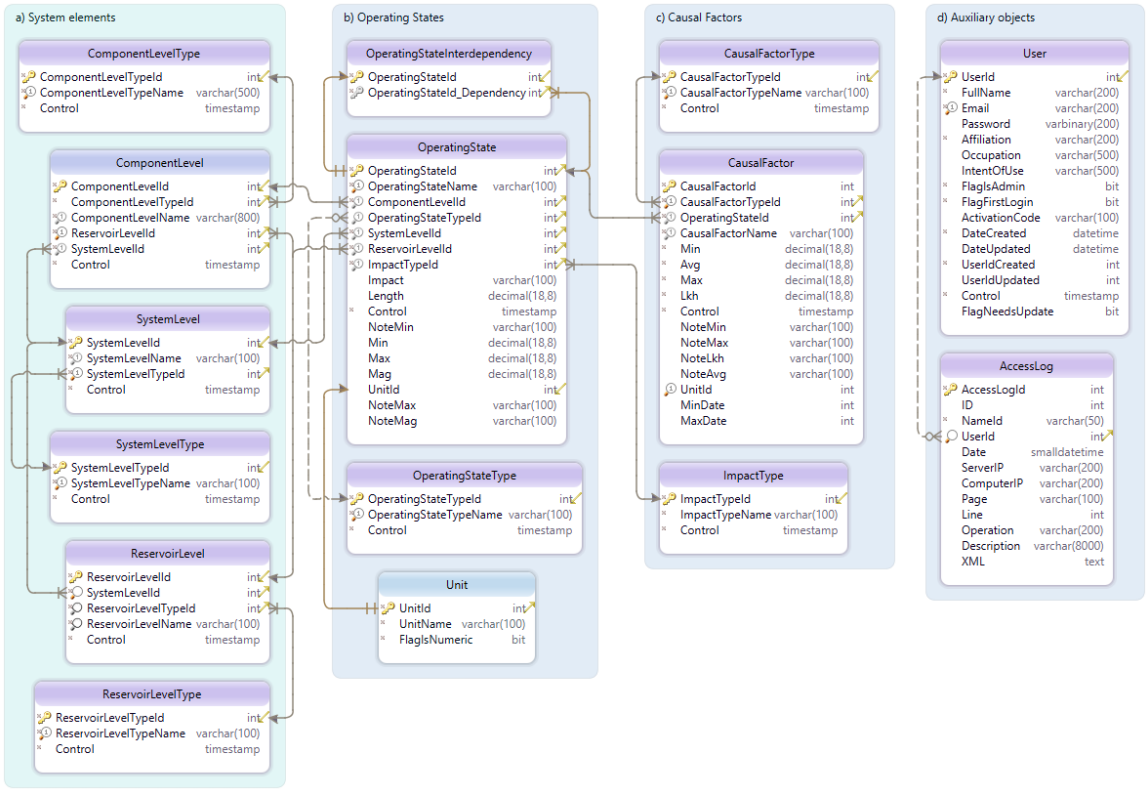


Figure 3-5: Database Structure

In order to keep track of and assess each individual component in the system, a hierarchical database structure is used in which components can be broken down into multiple sub-components and easily tracked using a components tree with drop-down lists containing components in an increasingly higher level of detail. Hydropower systems consist of a large number of complex, interacting components at various levels of the system, and often in

various locations. Use of a components tree helps the user set up the relationships between these higher and lower level components of the system.

The components tree consists of three levels. These are as follows:

- System level, which includes reservoirs, communications equipment and other high-level components of the overall system
- Reservoir level, which breaks down the system-level components into their sub-components. This includes dams, spillway gates, generating units, sensors and other infrastructure assemblies that exist for a single reservoir. Reservoir level components also include non-infrastructure system components such as the operations staff and inflow forecasting for the reservoir.
- Component level, which breaks down (when possible) the reservoir-level components into their sub-components. A spillway gate may have several interacting sub-components which function together, and these can be broken down at the component level and tied to the gate for which they represent.

These levels of the system are stored as tables in the database and can be seen in Figure 3-5a under “System elements” group and the object names are: “ComponentLevel”, “ReservoirLevel” and “SystemLevel”. The components tree ensures that each sub-component is tied to an individual component at a higher level, allowing for complex system structures to be represented fully while maintaining relationships between the higher-level components and the sub-components of which they are comprised. Components are also assigned a type to facilitate integration with the simulation model. Once components are defined, the individual operating states for each component can be described. The operating states database includes normal operating status for each component, as well as adverse operating states which can include failure or non-failure events. Operating states include several entries, shown in Figure 3-5b under “Operating States” group and listed below – these may be expanded upon, if necessary, for different systems being analyzed:

- Operating state type: normal, failed, failed closed, failed in place, collapsed, delayed, erroneous

- Impact type: none, outage, delay, error, blockage, settlement, cracking, wave, uncontrolled release of water
- Operating state description: qualitative descriptor for operating state
- Minimum impact: numerical minimum value of impact
- Maximum impact: numerical maximum value of impact
- Average impact: numerical average value (mode) of impact
- Unit type: The units of the impact
- Notes: user entries on data sources and/or assumptions

A numerical range of impact magnitudes is included (see "Operating States" table in Figure 3-5b since it may not be possible to estimate accurately the exact amount of time needed to repair certain components or the magnitude of error or delay which may occur under varying circumstances. For the more extreme failure modes, such as collapse of a dam or spillway gate, the process of repair could take years due to a number of factors including the degree of damage, the design process, the contract tendering process and even political considerations. Including a wide range of potential impacts for each operating state allows the full range of potential impact times to be explored. This structure also facilitates Monte-Carlo simulation which can be used to investigate the full range of system behaviour outcomes for a given scenario (set of component operating states).

Each operating state can be assigned one or more causal factors, with details as specified in Figure 3-5c under "Causal Factors". Causal factors are the events which lead to the operating state being described. Several causal factor types are required as various components of the system may be vulnerable to different disturbances. These include earthquake, maintenance, debris, excessive rainfall, ice, etc. The user may create a specific list for the system of interest. It may be desirable in some cases to define the magnitude of causal factor that could lead to the event. In the case of an earthquake, certain components may be vulnerable under certain degrees of ground acceleration. For some causal factors, it may not be possible to provide a numerical definition. Causal factors may also be assigned date index ranges, which specify the minimum or maximum date within which the causal factor can occur (as an integer between 1 and 365). An entry under causal factors was also added to allow for quantification of the probability of the causal factor leading to

the operating state. While this information is useful in overall calculations of scenario probability, it is extremely hard to define in the presence of limited data. The focus of this research is to define and analyze the full range of potential operating conditions for a dam system, and probabilistic assessment remains an important area for future work.

The procedure for population of the database is detailed in Figure 3-6. First, system documentation and details can be used to populate the components tree for the system of interest. Components at the different levels of the system are defined. Next, gathering of information relating to failures, expert knowledge, and any FMEA and STPA outcomes

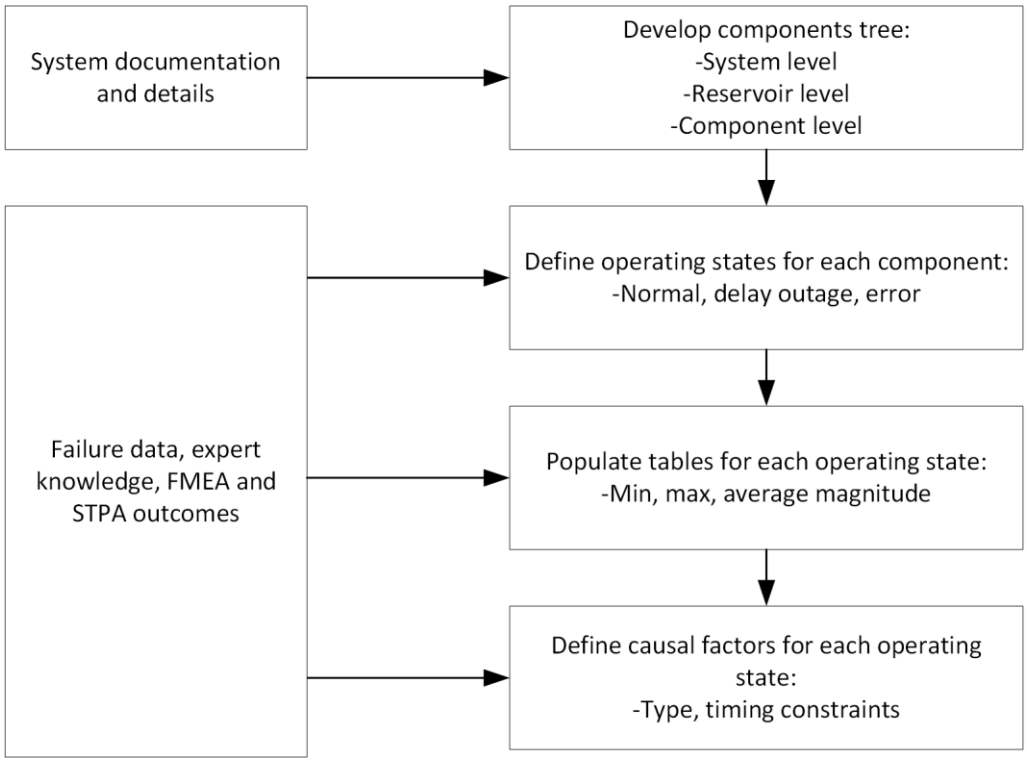


Figure 3-6: Component operating states database population flow chart

should be gathered to begin definition of operating states, population of operating states tables and causal factor information. Populating the table is quite similar to an FMEA in that it is expert knowledge and judgement from a variety of experts would be recommended in populating the database to ensure the most exhaustive list of operating states, impacts and causal factors which is as accurate as possible. The key difference from an FMEA is that non-failure operating states may also be included within the database. In addition,

component failure effects on other components or higher levels of the system can be programmed into the simulation model and do not need to be addressed within the database. While a significant amount of subjectivity remains in database population, placing the focus on individual components at different levels of the system and determining their direct impacts is necessary to allow for automated scenario generation. The database is structured so that various users can work together to provide inputs, facilitated using user identification (see "Auxiliary objects" in Figure 3-5d). Users of the database can enter in their details and create a user ID and password to be entered upon accessing the database interface. This, along with IP address tracking ensures all sensitive information is kept secure.

3.3 Operating State Scenario Development

The information in the database contains as many of the systems components and their potential operating states that can be defined by the modeller(s). A combinatorial procedure is required to automatically generate the complete list of operating state combinations from the database. Each scenario will represent one possible combination of operating states (or set of constraints) for each component in the system. The use of combinatorics will ensure an exhaustive list of potential operating scenarios is developed. Deterministic modelling of each of the scenarios, with Monte Carlo variation of their potential parameters, will allow for a more complete investigation of scenarios and potential system outcomes than may be possible using a purely stochastic model.

Consider a system of three components, **A**, **B**, and **C**, each which can either be functional or failed. The total number of possible combinations of operating states is $2^3 = 8$. These are:

$$ABC, \bar{A}BC, \overline{AB}C, \bar{A}B\bar{C}, \overline{ABC}\ A\bar{B}C, A\overline{BC}, AB\bar{C}$$

where a solid line over the variable represents its failed state. A process is required that can not only determine the number of combinations but can determine what the combinations themselves are. The process should also work for components with more than one

operating state, since this methodology considers operating states outside of the traditional binary definition of "functional" or "failed". The Cartesian Product meets these requirements. Consider a set of operating states for each component, such that:

$$A, \overline{A} \in \boldsymbol{A}$$
$$B. \bar{B} \in \boldsymbol{B}$$
$$C, \overline{C} \in \boldsymbol{C}$$

The Cartesian Product $\boldsymbol{A} \times \boldsymbol{B} \times \boldsymbol{C}$ defines all possible combinations, as listed above. Consider component $\boldsymbol{A}$ has an additional operating state in its operating state set, such that $A, \bar{A}, \hat{A} \in \boldsymbol{A}$. The operation $\boldsymbol{A} \times \boldsymbol{B} \times \boldsymbol{C}$ would then yield $3 \times 2 \times 2 = 12$ possible outcomes, as follows:

$$ABC, \bar{A}BC, \overline{AB}C, \bar{A}B\bar{C}, \overline{ABC}\ A\bar{B}C, A\overline{BC}, AB\bar{C}, \hat{A}BC, \hat{A}\bar{B}C, \hat{A}B\bar{C}, \hat{A}\overline{BC}$$

This example demonstrates the use of the Cartesian Product for generating all possible combinations of sets of varying lengths. To achieve the goal of coming up with all possible combinations of operating states for the system, the information from the database can be converted into operating state sets for each component of the system. Then, the Cartesian Product is applied to come up with a list of all possible combinations, where each combination is one unique set of operating states for each component in the system (a scenario). The process for scenario generation is detailed in Figure 3-7. The scenarios generated through this process become the input to the simulation model. The following paragraphs describe the steps and mathematical descriptions of the process.

From the database, tables detailing the database inputs at each level of the system can be extracted. The component number and operating state number for each of the components and operating states can be used to generate unique identifiers, as shown in Equation 3.1.

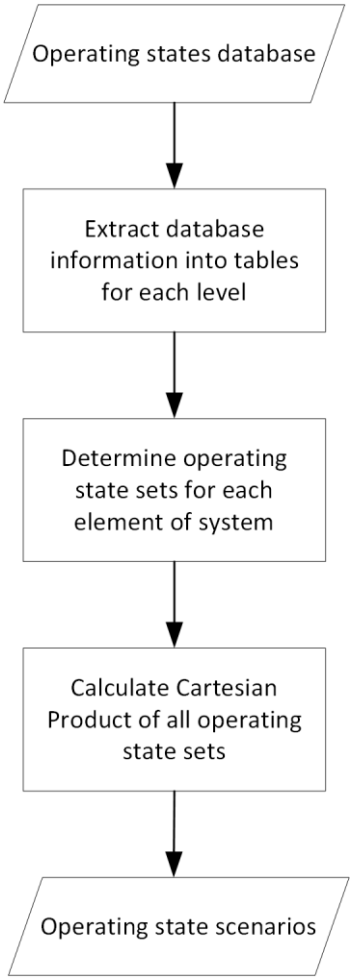


Figure 3-7: Scenario generation flow chart

$$\begin{aligned} C_1 &= [c_1os_1, c_1os_2, c_1os_3, c_1os_4 \dots c_1os_{m_1}] \\ C_2 &= [c_2os_1, c_2os_2, c_2os_3, c_2os_4 \dots c_2os_{m_2}] \\ &\vdots \\ C_n &= [c_nos_1, c_nos_2, c_nos_3, c_nos_4 \dots c_nos_{m_n}] \end{aligned} \tag{3.1}$$

where $C_1 \dots C_n$ represent the system component operating state sets for components 1 through n and $n \in (1 \dots N)$, $c_1 \dots c_n$ represent the components 1 through n, and $os_1 \dots os_{m_n}$ represent operating states 1 through m_n for component n which has m operating states, $m \in (1 \dots M)$. The component operating state sets contain a list of all unique operating state and causal factor combinations for a given component. The actual numbers given to components are generated directly from the database identifiers, and the operating states are then labelled 1 to m. The number of unique operating state/causal factor

combinations for each component may vary so the component operating state sets are not equal in length from component to component.

Cartesian product of these sets can be easily obtained using Python's *itertools* package, with the *product* function (Python Software Foundation 2012). The function is an efficient iterator containing nested "for" loops which essentially work as an odometer that advances the rightmost element on each iteration. This produces an exhaustive list of potential system operating scenarios, which contain one operating state for each component in the system. Using the Cartesian Product of each component's operating state set produces an exhaustive list of elements (scenarios) which include a complete list of operating states for every component in the system. Scenarios take the form shown in Equation 3.2:

$$S = [c_1 os_{m1}, c_2 os_{m2}, \dots, c_n os_{mn}] \tag{3.2}$$

where each scenario S consists of a single operating state for every component in the system. The operating states are kept track of using operating state identifiers as shown in Equation 1.

The total number of possible operating states *TS* is therefore equal to the number of elements in the Cartesian product of the component operating state sets. The number of elements in the Cartesian product is the product of the length of each set:

$$TS = \prod_{n=1}^{N} M_n \tag{3.3}$$

where M_n is equal to the number of individual operating state/causal factor combinations for each component $n, n \in (1 \dots N)$.

The resultant number of potential scenarios will be a function of the number of operating states and components, and as such will be extremely high for a complex system modelled in significant detail. Because the goal of this methodology is to simulate each and every scenario to determine the potential impacts, it may be necessary in practice to consolidate multiple components or causal factors into categories based on the modes of failure or adverse impacts, to reduce the number of scenarios and ensure the computational feasibility

of the simulation model. This could potentially be achieved through additional analysis – fault tree analysis may be particularly suited to determining multiple paths of failures that lead to the same higher-level fault, which could then be consolidated into a single operating state (this could be particularly promising for spillway gate and turbine systems).

While some of the generated scenarios may be relatively unrealistic in comparison to others, this approach focuses on determining all of the possibilities. The worst-case scenario where every component is in an undesirable state is extremely unlikely, yet still possible and does contribute (in a very small way) to the overall probability of failure. Understanding system behaviour in response to any scenario can help guide the selection of operating strategies and investments to improve system safety and guide system recovery.

This methodology for scenario generation does not consider the time between changes in operating states for different components or the inflows, which would significantly complicate the procedure. Instead, this is dealt with using a Deterministic Monte Carlo Simulation framework, where the operating states for each component (scenarios) are predetermined and used as inputs to a simulation model. For each deterministic simulation of a particular scenario, the uncertainty arising from varying time between events, impact magnitudes and inflows is varied in a number of Monte Carlo iterations, as detailed in Section 3.4.

3.4 Deterministic Monte Carlo Simulation Framework

This section presents a framework for the Deterministic Monte Carlo Simulation. First, a description of the simulation model development is provided. In this research, a system dynamics simulation environment is used. Next, a discussion of the Monte-Carlo variation of scenario parameters is provided. The system dynamics simulation model is run in a hybrid deterministic/Monte Carlo environment where (a) the operating states associated with a single scenario are used as inputs for a single execution of the simulation, and (b) the parameters of that particular scenario are varied in a series of iterations, using Monte

Carlo generated parameters for operating state timing, impacts and inflows. The final section of this chapter details the simulation execution steps.

3.4.1 System Dynamics Simulation Model Development

Simonovic (2009) presents simulation techniques that deal with water resources in general, with a particular focus on system dynamics simulation as a tool for water resources engineers. In system dynamics, a stock-and-flow model can be used to represent the system structure, showing the complex interactions between system components. These complex interactions are the source of the system behaviour and can help identify emergent behaviours that may not be easily assessed through analysis of the system's individual parts. By modelling the system as a whole and all relevant feedbacks and relationships between components, the overall system behaviour can be characterized. The stock-and-flow representation facilitates easy modification of the system structure to experiment with various upgrades or operational strategies that have the potential to improve system performance. Recall, stocks are represented as boxes and flows are represented as pipelines into or out of the stock controlled by spigots (with a "source" or "sink" that supplies or drains flows). Flows have units of material over time, and while inflows and outflows for the dam system are represented as flows in this particular application, there are many other types of flows that can be used which may have nothing to do with water. Auxiliary variables and arrows make up the other major components of a stock-and-flow model, and these represent constants or variables that change with time according to a mathematical equation or algorithm.

Consider a simple dam system, with a single reservoir and dam that is controlled only by a free overflow weir. Figure 3-8 contains a representation of this system. Reservoir Storage is represented as a stock with units m^3. The value of Reservoir Storage can only be changed by the flows into or out of the stock. Flows have the same units as the stock over time. In this example. Inflow and Outflow are the stocks and have units of m^3/s.

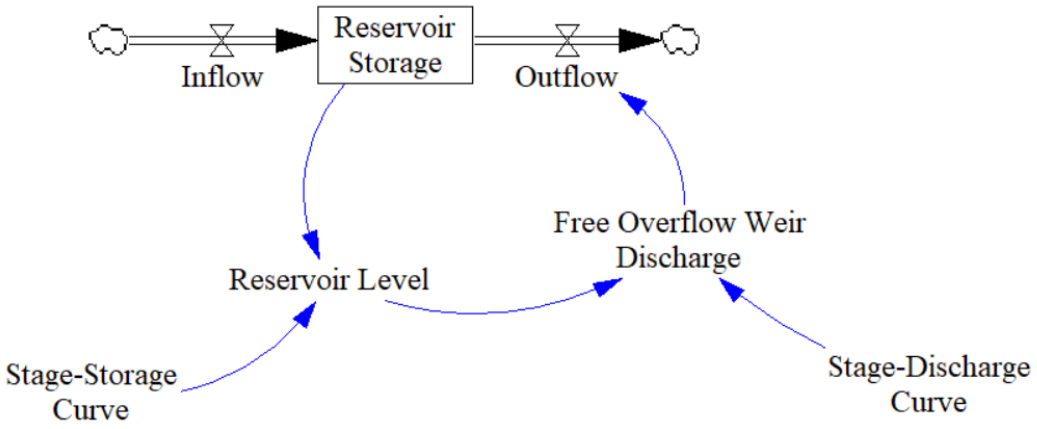


Figure 3-8: Simple dam system with free overflow weir

The change in Reservoir Storage can be computed as:

$$\frac{dS}{dt} = I - Q_{out} \tag{3.4}$$

Where S represents Reservoir Storage, t represents time, I represents Inflows and O represents Outflows. System dynamics tools use integration to calculate the value of the stock at each timestep. The Reservoir Level (RSE) value is a function of the Reservoir Storage as defined by the Stage-Storage Curve. Similarly, Free Overflow Weir Discharge (Q_{weir}) is a function of the Reservoir Level, as defined by the Stage-Discharge Curve for the weir. Equations presented here are generated from the Cheakamus System characteristics, the details of which are summarized in Appendix A.

$$RSE = -1.12 * 10^{-5} * S^2 + 3.24 * 10^{-2} * S + 3.64 * 10^2 \tag{3.5}$$

$$Q_{weir} = -35.8 * RSE^3 + 40.9 * 10^3 * RSE^2 - 15.6 * 10^6 * RSE + 19.8 * 10^8$$
$$Q_{weir} = 0 \; for \; RSE < 378.41 \tag{3.6}$$

Note that for "flashy" reservoirs with little storage in comparison to inflow volumes, the reservoir elevation may vary greatly throughout the day, so weir discharges may also vary hourly. For a model run on a daily timestep, it may be necessary to compute the Free Overflow Weir Discharge, Q_{weir} for a given day by iterating within the function over a 24-hour period. This will ensure weir discharges accurately reflect reality.

Inflows represent an external model input, which in this case ranges from 5 to 25 m^3/s. Outflows (Q_{out}) are equal to the Free Overflow Weir Discharge:

$$Q_{out} = Q_{weir} \tag{3.7}$$

If we simulate this model with a constant inflow and user-defined Stage-Storage and Stage-Discharge curves, as well as an initial starting reservoir level (10 m^3/s-d), we can see that the model reaches a steady state, where the discharge over the weir is equal to the constant inflow. This is shown in Figure 3-9 for three different constant discharge values. Note that the reservoir level can be represented using units of volume (m^3), however using the units m^3/s-d can considers the available flow rate over time, which simplifies the calculations required.

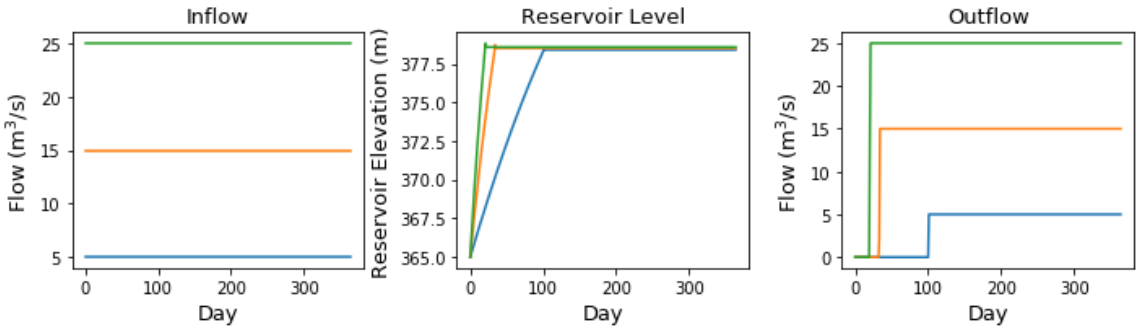


Figure 3-9: Simulation of simple dam system with free overflow weir

Suppose we add a single gate to this system, modifying the stock-and-flow diagram in Figure 3-10. The Gate Discharge (Q_{gate}) is a function of the Gate Position (GP) and Reservoir Level, as defined by the Gate Rating Curve (provided by BC Hydro and summarized in Appendix A). For this single-gate system, the rating curves for both gates are combined into a single gate (by simply adding the discharge columns). The gate rating curves can be used in a two-dimensional interpolation to determine what the corresponding flow is for a given reservoir elevation and gate position. This can be done using simple Python functions such as *interp2d* which is part of the *scipy* package.

$$Q_{gate} = f(RSE, GP) \tag{3.8}$$

The outflow then becomes:

$$Q_{out} = Q_{weir} + Q_{gate} \tag{3.9}$$

Simulating this system for constant inflows of 32 m^3/s with a variable gate position yields a similar result where a steady state is achieved, as shown in Figure 3-11. In the blue line in Figure 11, the gate position is the smallest and the reservoir rises to the free overflow spill. At this point, an instantaneous increase in the outflow is observed as the overflow spillway begins to pass flow. It is important to note that the increase in outflow at a smaller time step (say, hourly) would be more gradual than the daily plots may indicate.

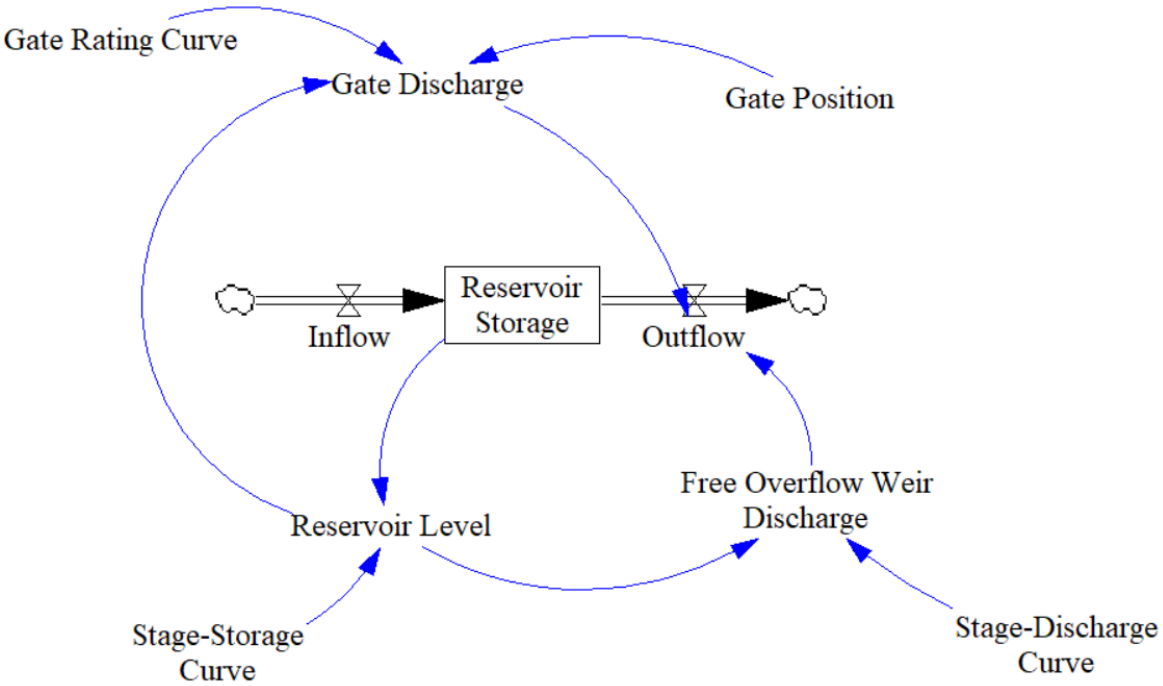


Figure 3-10: Simple dam system with free overflow weir and gate

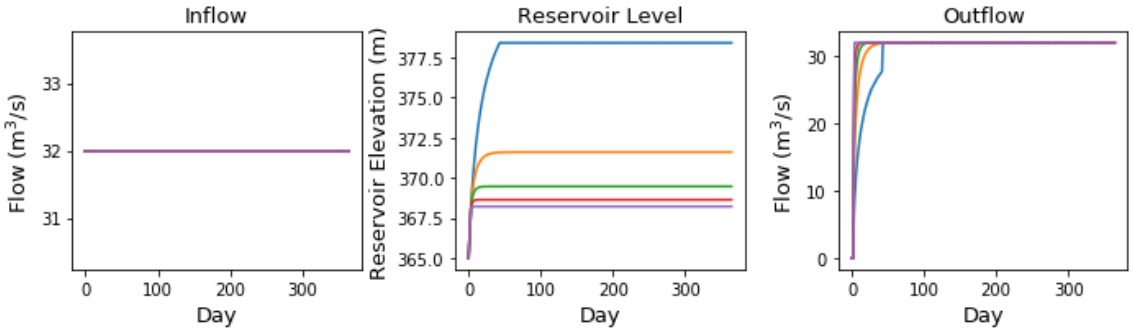


Figure 3-11: Simulation results for simple system with free overflow weir and gate

If we add a sinusoidal relationship to represent seasonal variations in the system inflow, we start to see some more variation in the resultant reservoir levels, as shown in Figure 3-12. The relationship used in this simple simulation model is chosen to roughly mimic the natural seasonal variation in flows for the Cheakamus system, with time t:

$$I = 60 * \sin 0.015(t - 80) + 70 \qquad (3.10)$$

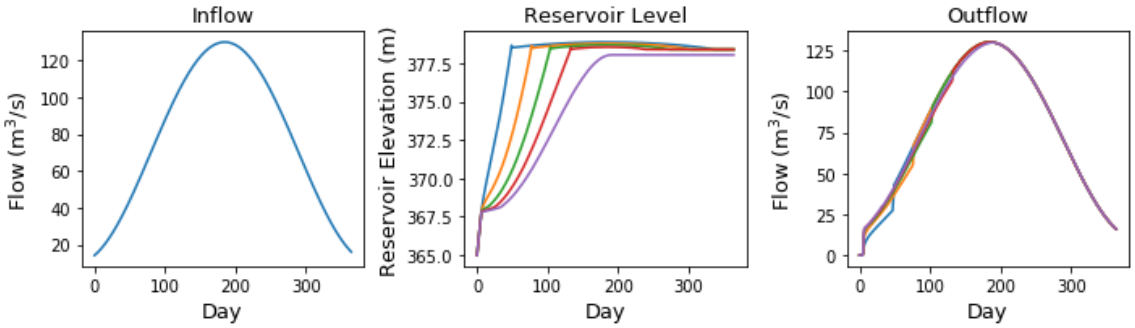


Figure 3-12: Simulation results for simple system with free overflow weir, gate, and sinusoidal-varying inflows

Including some day-to-day variability in inflows for the simple system can be done using a random normally distributed variable (with a mean of 0 and a scale of 30) which is added to the time-dependent sinusoidal inflow value. The value is then truncated, so that the minimum inflow value cannot be less than 2 m^3/s. The simple system simulation results with added daily variability are shown in Figure 3-13. The fluctuation in the reservoir levels is more extreme, and the resulting reservoir levels and inflows begin to vary more greatly.

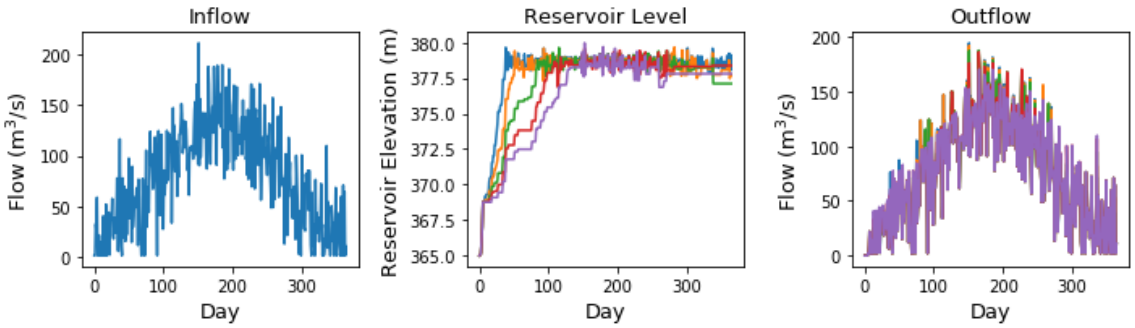


Figure 3-13: Simulation results for simple system with free overflow weir, gate, sinusoidal inflows with daily variability

Obviously, for normal operation of a dam with a gated spillway, the gate positions are not kept constant throughout the year and for all inflows. Gate positions may be selected based on a number of inputs, including the inflow forecast, rule curves, target reservoir elevations, outflow constraints and downstream impacts. Creating an algorithm to simulate operations planning is a more challenging aspect of model development, in particular in the case of cascading and parallel dam systems. Optimization is frequently cited in the literature and works well for balancing inflows, downstream effects, reservoir operational limits and outflow constraints. However, optimization may significantly impede computational efficiency, which is an important consideration when running the simulation model a large number of times. For this simplified example based on a version of the Cheakamus project with only a single gate, an if-then-else type algorithm has been developed for Operations Planning. The algorithm uses 14 days of inflow forecasting to determine the appropriate gate releases that will keep the reservoir level between target elevations (see Figure A5 in Appendix A). Inflow Forecast is based on the sinusoidal relationship described previously, with no random normal variable added. This means the operations planning algorithm has some indication of the average inflow to expect over the next 14 days but is not aware of any major deviations from the normal level due to the random normally distributed variable that is added. The Operations Planning algorithm is detailed in Figure 3-14 (King and Simonovic 2020). The output variable from Operations Planning for this simple example is gate flow, which can be transformed into a gate position instruction. The Gate Position is calculated using a reverse two-dimensional interpolation

using the Operations Planning output (the desired gate flow) and the Reservoir Level, based on the Gate Rating Curve. The resultant stock and flow model is shown in Figure 3-15.

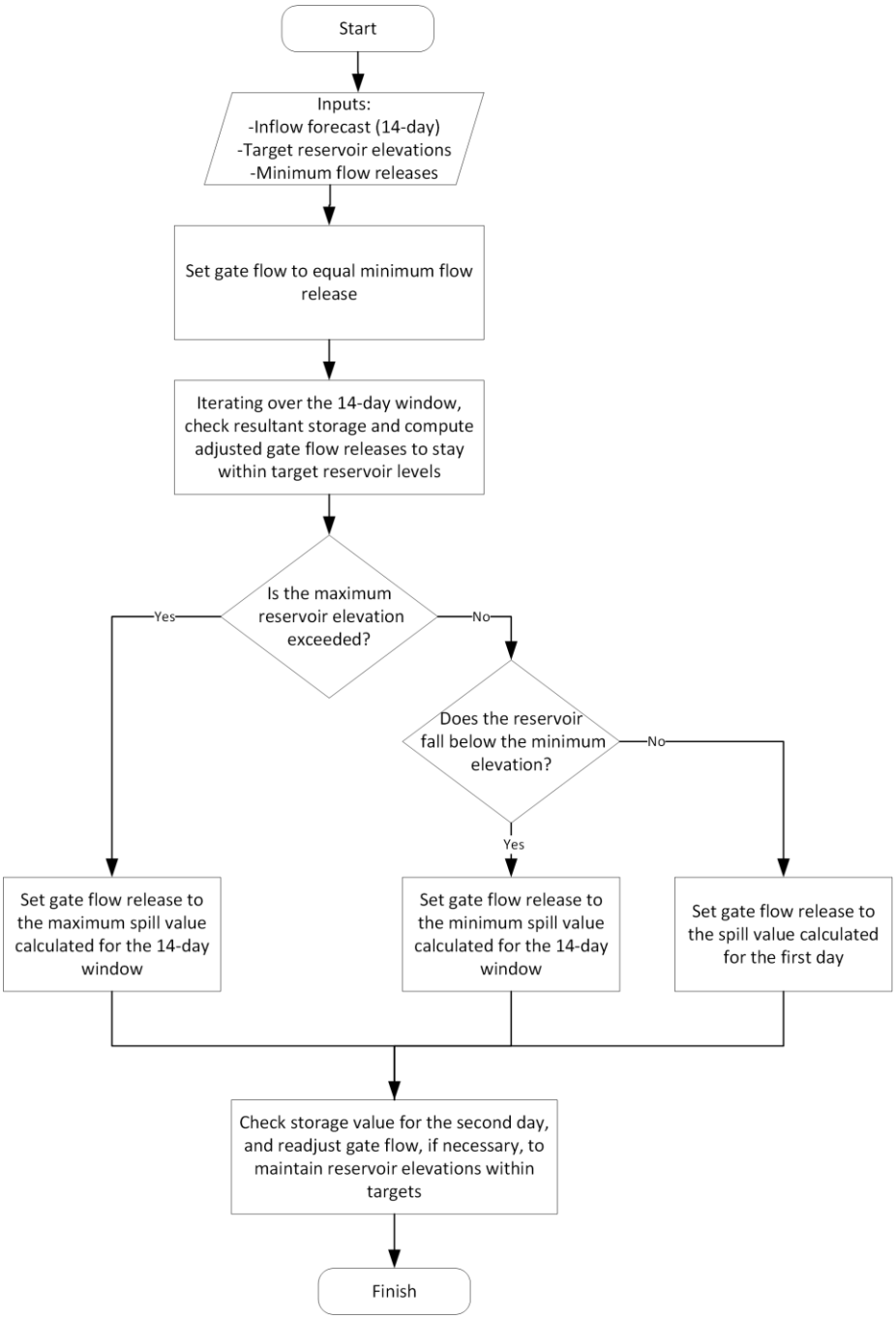


Figure 3-14: Simple operations planning algorithm King and Simonovic (2020)

The simulation results for the example with operations planning implemented are shown in Figure 3-16. The reservoir rises to the level of El 376.5 m (the Normal Maximum Reservoir Level), and hovers at that level or just above and below based on the deviations introduced by the random normal variable added to the inflow. Note that there are no power flow release facilities included in the model, so the algorithm keeps the reservoir level high because there is no reason to discharge more additional water than necessary to meet the maximum reservoir level target.

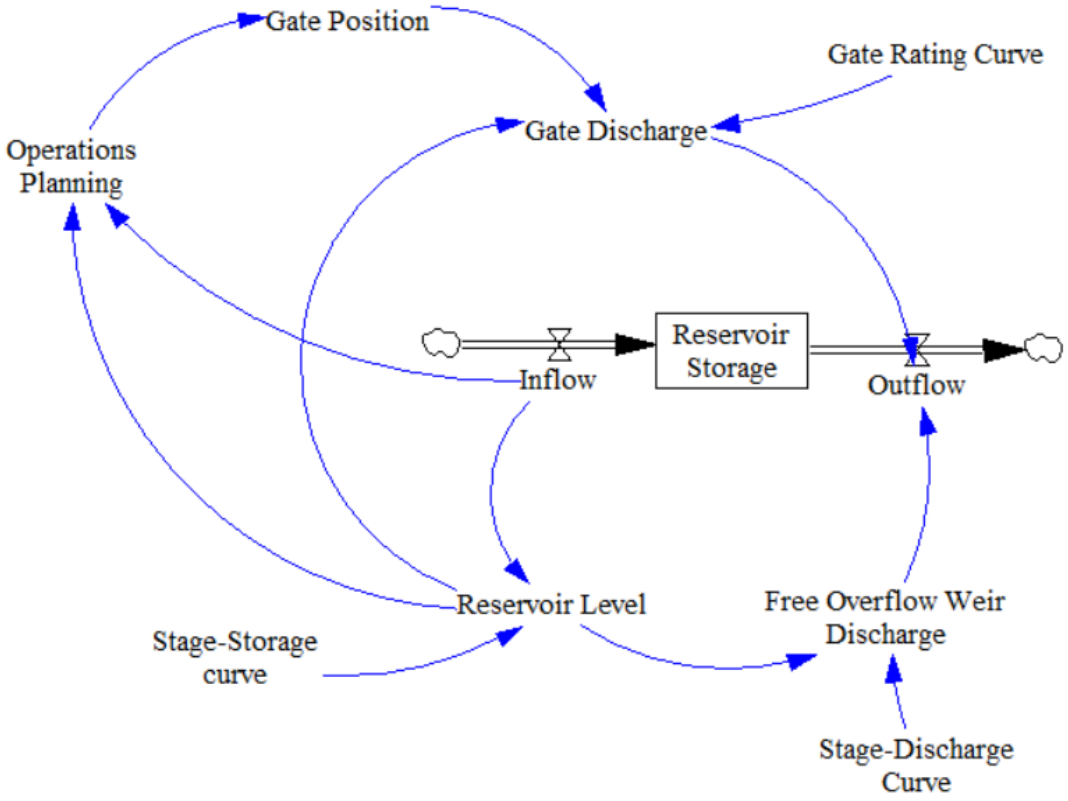


Figure 3-15: Simple dam system with a single weir and gate, with operations planning algorithm implemented (King and Simonovic, 2020)

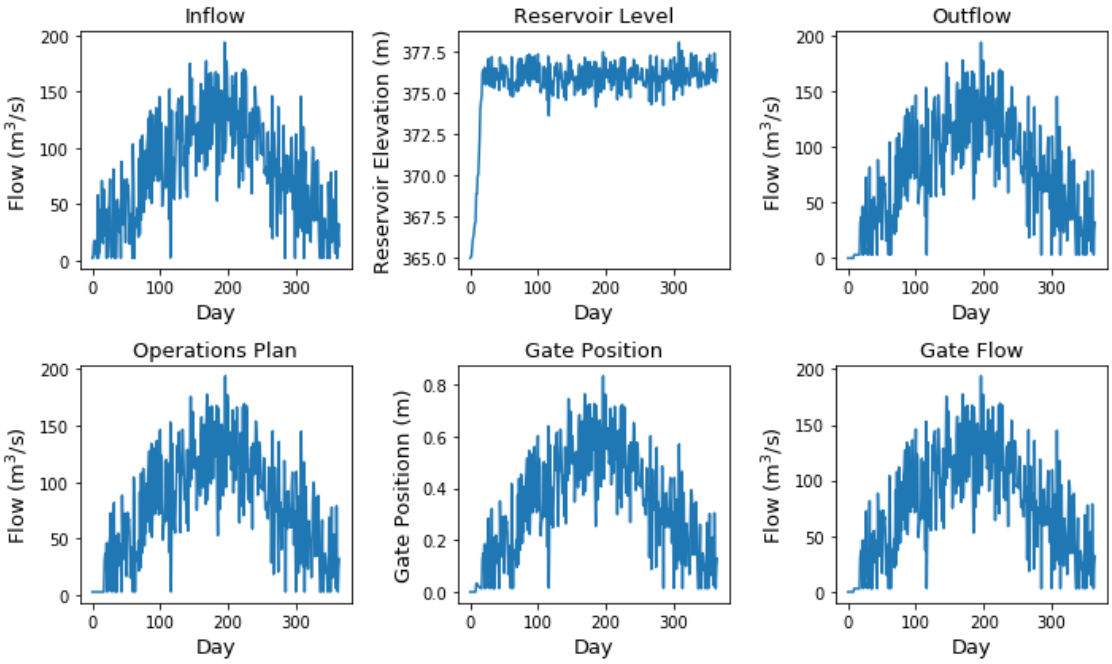


Figure 3-16: Simulation results for simple dam system with single weir and gate, with operations planning algorithm implemented (King and Simonovic, 2020)

Another important feature of the simulation model will be the ability to simulate component failures or outages. Considering the simple example developed, this can be added by creating a variable that tracks remaining time to repair following gate failures, Gate Remaining Time to Repair, $GRTTR$. This is modelled as a stock, which receives a pulse of Gate Failure, GF, when the gate fails. The stock drains with the value time when its value is positive, using the flow Gate Repair, GR. The gate remaining time to repair can then be implemented in the model based on the impacts of a gate outage – in this simple example the situation modelled will be that the gate fails in the closed position. Gate failure causes an inflow to the stock of 20 days at time $t = 100$, and the gate becomes stuck in the closed position for a 20-day period. The modified stock and flow diagram for this is shown in Figure 3-17.

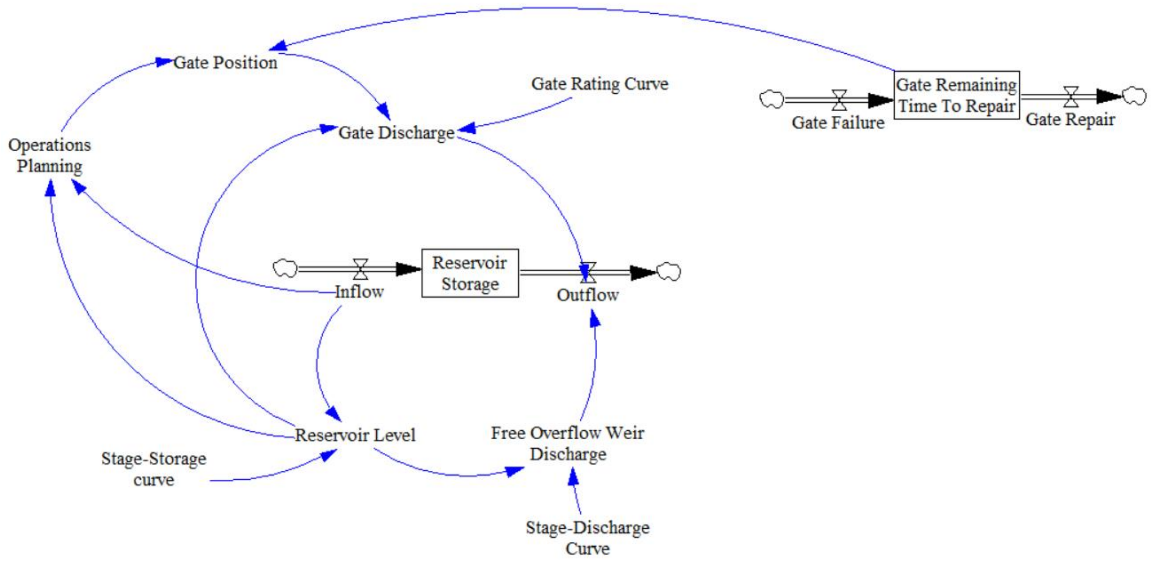


Figure 3-17: Simple system with gate and weir, with operations planning and gate failures implemented (King and Simonovic 2020)

In this example, the gate remaining time to repair is calculated as:

$$\frac{dGRTTR}{dt} = GF - GR \tag{3.11}$$

Gate Failure is calculated as:

$$if\ t = 100, GF = 20$$

$$else, GF = 0 \tag{3.12}$$

Gate Repair is calculated as:

$$if\ GRTTR > 0, GR = 1$$

$$else\ GR = 0 \tag{3.13}$$

The Gate Position can then be calculated based on the gate availability:

$$if\ GRTTR = 0, GP = f(OP, RSE)$$

$$else, GP = 0 \quad (3.14)$$

where OP represents the operations planning output, which is representative of the desired gate flow. Simulating this model yields the outcomes shown in Figure 3-18. The impacts of the gate failure can be seen in the image starting at day 100 of the simulation, where the gate position and gate flow drop to zero, and the reservoir elevation rises above the target elevation. Flow over the free overflow weir is observed during the gate outage (these are not shown but are the difference between Outflow and Gate Flow). Once the gate is back online, the gated spillway flow is increased significantly to reduce the reservoir elevation to within the target levels. The inflow on the day of the gate's return to service is less than predicted, so the operator opens the gate more than is necessary and the reservoir drops to just above the gate sill elevation (El. 367.28 m). In reality, operators will have a relatively better idea with respect to the expected inflow. Dam operators would also be able to adjust the gate position within the 24-hour period if the inflows are less than predicted to ensure rapid drawdown of the reservoir does not occur. This is one potential limitation of running the model on a 24-hour timestep, though improved inflow prediction for the operations planning algorithm would avoid the issue. For less flashy reservoirs, a daily time step may be adequate.

As more features are added to the simple simulation model, the nonlinearity of the problem becomes more obvious. Calculation of the reservoir level response becomes increasingly more complex as additional components, variable gate positions, natural variability in inflows, outages, etc. are added to the simulation model. These interactions are not easily modelled using traditional tools, so simulation is necessary for quantifying the system response to various inputs. These simple examples help build a clear case for system dynamics simulation as a tool to determine reservoir level response to various operating conditions. The system dynamics platform offers a particularly suitable modelling environment for complex, dynamic systems with interactions among components. The object-oriented building blocks help visualize the connections between the different components of the system. This visual representation of the system structure can be inspected to help gain confidence in the model performance. Subscripting is another useful

tool that can help modellers easily add series and parallel dams to a system without complete re-programming. Subscripting is also useful for the modelling of multiple sub-components or redundant features of the system.

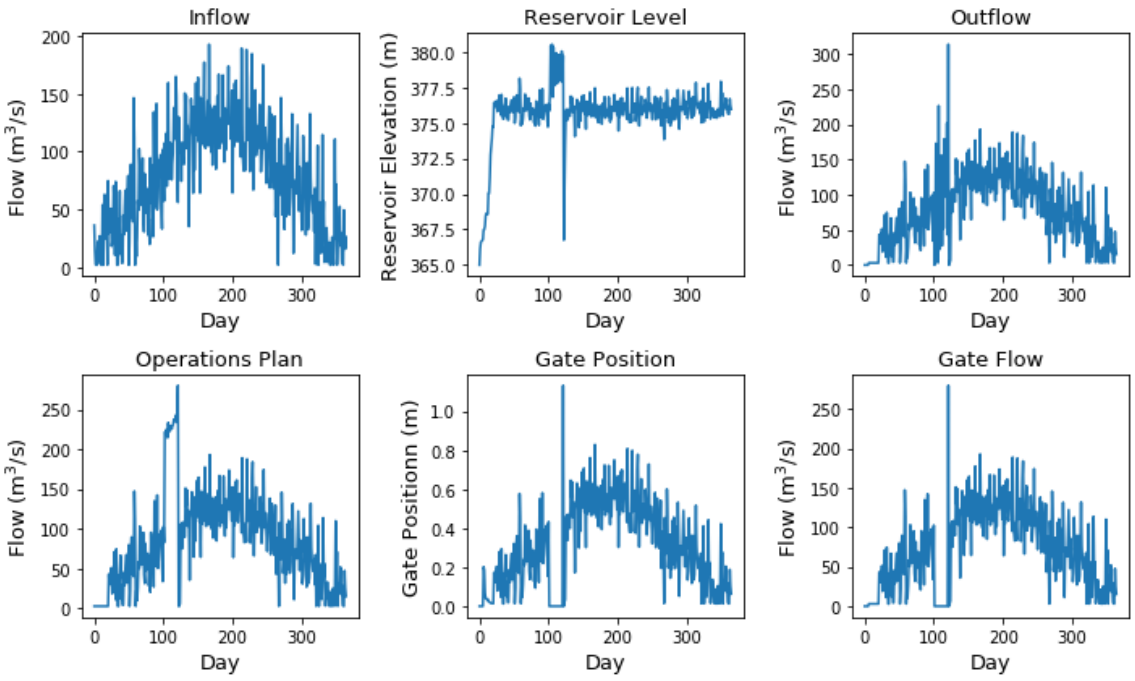


Figure 3-18: Simulation of simple system with single gate and weir, operations planning, and gate failure implemented (King and Simonovic 2020)

The general process for the development of a system dynamics simulation model for a hydropower system is described in Figure 3-19 (King and Simonovic, 2020). The process is iterative, that is, model development is influenced by model testing, and development continues until the modeller is satisfied that the modelled system is an adequate representation of reality. The model is a description of physical and nonphysical relationships among system components. A significant amount of information about the system is required to define these relationships mathematically, and expert judgement is necessary in model development. Dividing the model into interconnected sub-systems shown in different views or sectors may be helpful to organize the model presentation. Sub-systems may be connected to each other by one or more variables. These sectors can follow a generic control loop, such as the one presented in Section 1.3 and Figure 1-20, as described by Leveson (2011) and adapted for a hydropower system. The sectors include:

(1) A controller, who interprets information relating to the state of the system and produces a set of operating instructions, (2) Actuators, the mechanical-electrical assemblies which work to move gates in the controlled process, (3) A controlled process, representing the infrastructure being controlled or the hydraulic system state, (4) Sensors, which relay information back to the controller and (5) Disturbances, which are not directly part of the control loop but may affect the functionality of any one of its features. This high-level system structure represents a hierarchical system of systems, with each box representing its own system (Leveson 2011). King et al. (2017) presents a detailed system dynamics model of a dam system that is broken down into control system sectors. The benefit of developing a detailed model of the system components is that low-level failures and other events within the system can be initiated and the simulation model can determine the system-level impacts for a particular set of inflows and event parameters.

Selection of the variables that will be required to adequately represent the system is another important step. The variables represent states of the system which the modeller is interested in over time, and there may also be a number of intermediate variables that transform information between the key variables of interest. The variable types are stocks, flows and auxiliary variables. Stocks may include reservoir levels, remaining repair times and even gate positions, depending on how the modeller wants to set up the equations. Flows are the values which have units over time and represent the inflow and outflow of the stock value. Auxiliary variables are neither stocks, nor flows, and may represent physical or nonphysical relationships and processes. Subscripting can be used so that a single representation of a variable and its relationships (equations) can be applied. This is particularly useful for representation of multiple reservoirs, gates or other redundant features of the system. Defining the relationships requires expert knowledge of the system, data, and programming capability. Some auxiliary variable equations can be represented by simple if-then-else type formulae, others may represent nonlinear relationships or even complex algorithms with a number of processes occurring internally.

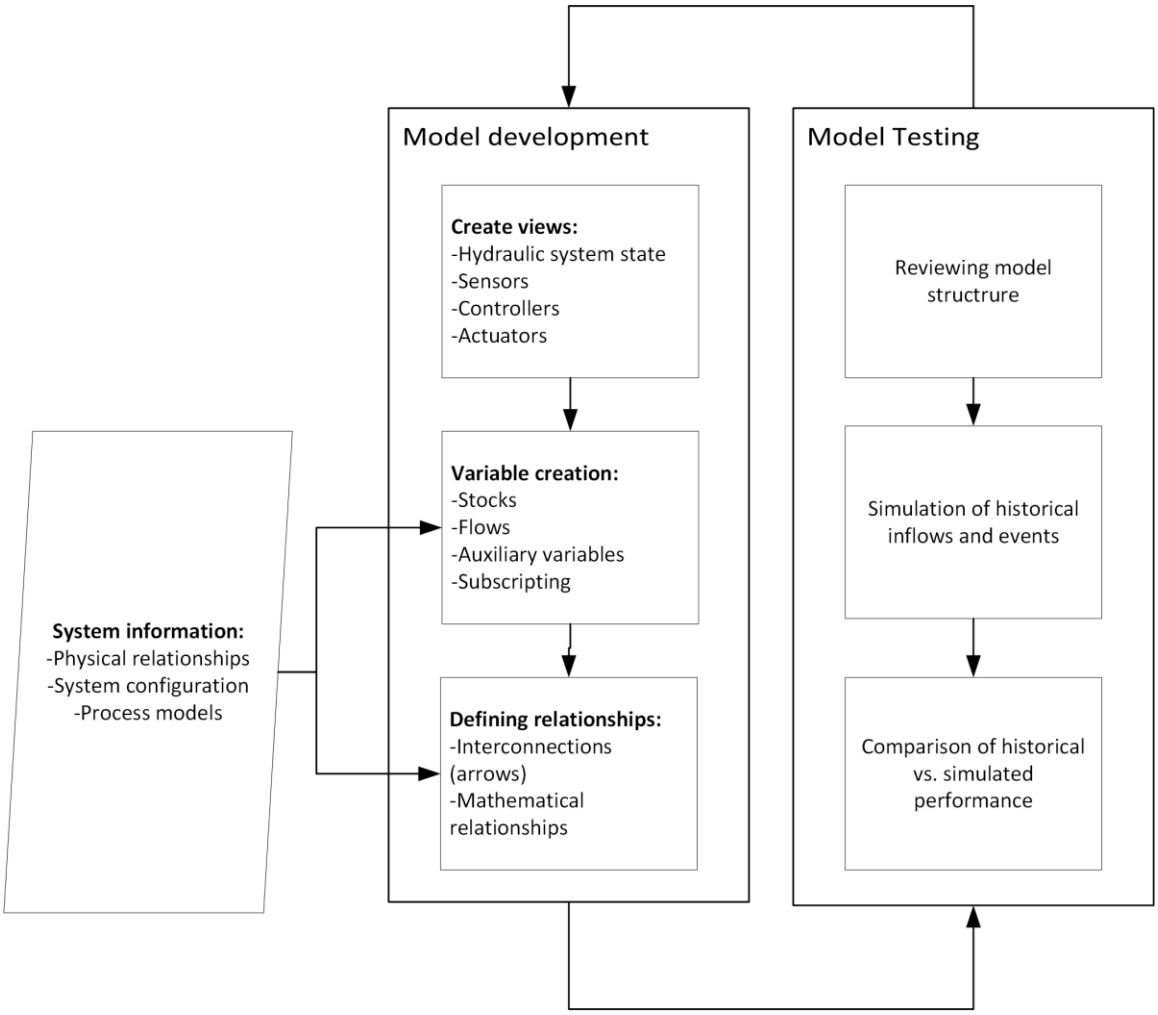


Figure 3-19: Simulation model development flow chart (King and Simonovic 2020)

The model output is only as good as the modellers understanding of the interactions and relationships within the system being analyzed. Like all models, simulation models are abstractions of reality. Sterman (2000) argues that, because of this, all models are "wrong" and that simulation models can never be validated or verified in the traditional sense of the word. There are, however, a number of tests that can be done to gain confidence in the model performance. These tests should be done iteratively throughout the model development process. Analyzing the system structure and feedbacks to ensure all important variables are represented and their equations are grounded in reality is important. This includes checking the water balance, rating curves and other physically-derived variables. Checking the dimensions is another important model test. Historical records of system

operation are also particularly useful for testing and development of the model. A direct comparison between simulated and actual values provides information to the modeller about how well the system is mimicking reality in terms of normal operation. Comparison of system reservoir levels and outflows can help the modeller adjust the system structure so the system behaviour better captures the dynamics – this is particularly important during the development of operating rules. Once the model results are relatively close to reality, the model is ready for simulation.

3.4.2 Monte-Carlo variation of scenario parameters

Each automatically generated scenario (see Section 3.3) contains a list of component operating states which may be normal, erroneous, failed, etc. Each component operating state is tied to one or more causal factors and has a specified range of impacts that can be expected should the operating state occur. Impacts may include outage length, error magnitude, or delay length. Linking this information into the simulation model in a way that allows a wide range of potential outcomes to be explored for each scenario is a critical part of the implementation. An example of such a link was shown in the previous section. System dynamics modelling is inherently deterministic, so specific instructions for how to implement the scenario must be given to the model before running. Monte-Carlo selection of simulation inputs is considered to be the most efficient way to assess the outcomes from as many implementation possibilities for a single scenario as can be achieved within the computational time constraints. Each operating state has varying impact magnitudes between minimum and maximum values specified in the database. In addition to this, the adverse operating states may be occurring within some temporal proximity to one another but not at the exact same time. Inflows may also significantly affect the way an operating state changes the system behaviour. While simulation facilitates the assessment of component interactions, feedbacks and nonlinear system behaviour, the Monte-Carlo variation of these important simulation inputs can help better capture a range of system behaviour that is possible as a result of a given scenario. The temporal proximity of the adverse operating states, the magnitude of impacts and the system inflows can all easily be varied using a Monte-Carlo simulation approach. Each scenario can be run many times

(iterations), varying these inputs to explore the system behavior in response to a random subset of the total implementation possibilities for each scenario. This helps provide more information about the uncertainty associated with a particular scenario in terms of the range of system response that can be expected.

A wide range of hydrological conditions may be tested for each operating scenario, by selecting a random year and start date for each Monte-Carlo run of each scenario. The year and start date can be linked to a synthetic inflow time series (this is discussed in Section 3.5.2.

Operating state impact magnitudes may be difficult to estimate, and can vary significantly depending on the timing, organizational factors, and availability of materials to rectify the adverse operating state, etc. The potential range of operating state impact magnitudes is represented using minimum impact, maximum impact and average impact (mode) as specified in the component operating states database in the operating states description. This information can be used to generate Monte-Carlo inputs with a triangular distribution (Kotz and van Dorp 2004):

$$Impact = \begin{cases} i_{min} + \sqrt{U(i_{max} - i_{\min})(i_{avg} - i_{min})} & for\ 0 < U < F(i_{avg}) \\ i_{max} - \sqrt{(1-U)(i_{max} - i_{\min})(i_{max} - i_{avg})} & for\ F(i_{avg}) \leq U < 1 \end{cases} \quad (3.15)$$

where U represents a random variate from the uniform distribution between 0 and 1, i_{min} represents the minimum impact value specified in the database, i_{max} represents the maximum impact value specified in the database, i_{avg} represents the average value specified in the database and $F(i_{avg}) = (i_{avg} - i_{min})/(i_{max} - i_{\min})$. A random impact magnitude for each operating state is generated in this way and used as the second Monte-Carlo input to the simulation model.

Timing of events may also vary within a scenario, and events can occur at the same time or within hours, weeks or even months of one another. The temporal proximity of events represents the third Monte-Carlo input to the simulation model. The causal factors for each operating state play a roll in determining operating state proximity. The number of causal

factors can be used to determine the number of time steps between adverse operating states arising from different causal factors. Operating states with the same causal factor (for example, an earthquake) are initiated at the same time. Operating states for subsequent causal factors are initiated at some value, Δt_{nc} in the future where $nc \in (0, \ldots, NC)$ and NC is equal to the number of unique causal factors less one (because the first causal factor is implemented at time t=0 in the simulation). The ordering of causal factors is also shuffled for each iteration so that the first operating state(s) change between Monte-Carlo inputs. For some causal factors, including lack of maintenance and aging, impact timing is completely randomized if more than one component is affected; that is, failure of one component due to lack of maintenance may not occur at the same time as the failure of another component that has not been maintained. There may be a time limit within which these events can occur, as defined by the user for the system of interest. This is a parameter that helps increase the chance that the events are impacting one another so that the scenarios represented in the outputs are reflective of the input scenario (discussed further in Section 3.4.3).

In addition to generating these randomized parameters for each scenario, it is necessary to program component-specific connections that link the database's operating state identifiers to the specific point in the simulation model where the component failure, error or delay occurs. An example of how this can be done was provided in Section 3.4.1. The timing and impact magnitude can be represented by variables that change with each Monte Carlo iteration. Inflow sequences for each iteration can be selected from the historical record using the randomly generated start day and year. Directing the impact towards the correct component and implementing it requires significant modelling effort and expert judgement. The implementation of these connections will differ from application to application and must be done at the front-end of the simulation model to ensure the scenario information is routed properly through the simulation model. The randomly generated impact parameters and timing must be connected to the appropriate variable within the simulation model. Once the connections are made, simulation can proceed following Figure 3-20 as discussed in the following section.

It is important to note that random numbers generated by computer code are not truly random, because they rely on algorithms that repeat. They are technically “pseudo-random” numbers. When a very large number of scenarios is run for many iterations, there is a possibility that a pattern may be observed within the random numbers generated by the model. This issue is not explored further in this research but remains an important issue in computational science.

3.4.3 Deterministic Monte Carlo Simulation Process

The process for scenario simulation is described in Figure 3-20 (King and Simonovic, 2020). In the simulation, each scenario is given a unique simulation number (“seed number”). At the start of the simulation, a “seeds to run” list is developed. Each seed number corresponds to a line in a list of the scenarios, which contains a unique set of operating state combinations for the system. This is used to gather the information from the database tables and set-up the Monte-Carlo parameters for the particular scenario being simulated. The Monte Carlo parameters are randomized inputs that vary within the bounds specified in the operating states database. This allows for a more subset of the potential outcomes for a given scenario to be explored. Once the Monte Carlo input generation is completed based on the scenario of interest, the simulation of the scenario proceeds.

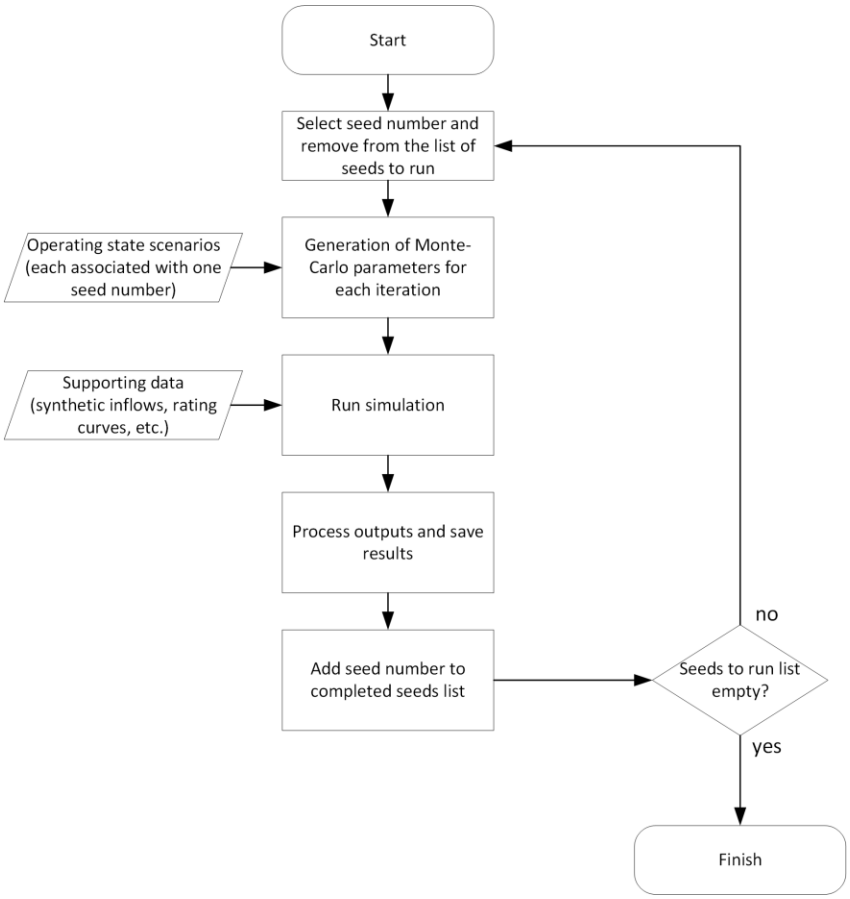


Figure 3-20: Simulation flow chart

Following the simulation of each scenario iteration, timing considerations must be addressed, to ensure the results are accurately attributed to the scenario being represented. This can be done by analyzing the "system state deviance" to determine whether subsequent events are dependant on preceding events. An event dependency algorithm to analyze the outputs from each iteration is necessary in order to count the simulation results towards the scenarios that are truly represented within the output data. Recall the example reservoir elevation plots for two iterations shown in Figure 3-21 (King and Simonovic, 2020). Given the time of occurrence of A, B and C, the reservoir level under normal operations, and the resultant reservoir levels, a simple comparison can be used to determine whether events are influencing one another. The algorithm to analyze scenario outcomes is shown in Figure 3-22 (King and Simonovic, 2020).

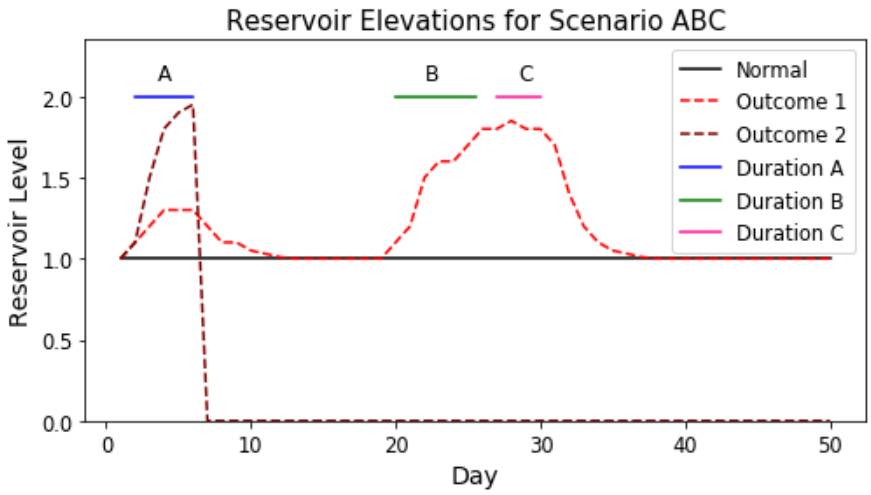


Figure 3-21: Example output reservoir elevations for Scenario ABC (King and Simonovic 2020)

First, an empty event list is created, and the time is set to $t = 0$. The analysis starts by first checking if a new event (adverse operating state) is initiated at the current time step The event initiation time is determined through Monte-Carlo sampling. If a new event is initiated, the event is added to the event list. If no previous events are in the list, time *t* represents the scenario start day. If there are events in the event list, a check is done to see whether the event impacts are over – this is a simple comparison of the following *y* days of simulated reservoir elevations with the previously expected reservoir elevations for that set of inflows. The choice of the number of subsequent days to be compared depends on the system being modelled and may be shorter or longer depending on the storage relative to the inflows. If the elevations are within a certain threshold, x, of the previously expected reservoir levels for all days within a three-day period, the scenario is considered to be over. The threshold is a small number that indicates the reservoir levels are basically the same – it may also vary depending on the reservoir being modelled and must be chosen by the analyst for the system of interest. Once the reservoir levels are restored to the previously expected values, the results for the scenario are saved, the event list is emptied, and the analysis proceeds to the next timestep. If the elevations are not yet matching, the analysis proceeds to the next time step, as long as the reservoir elevations haven't risen to a

sufficient level to fail the dam by overtopping. If the dam has failed by overtopping, the results are processed and saved for the events in the list. The process continues through all of the timesteps, until either there are no more time steps to analyze or the dam has failed.

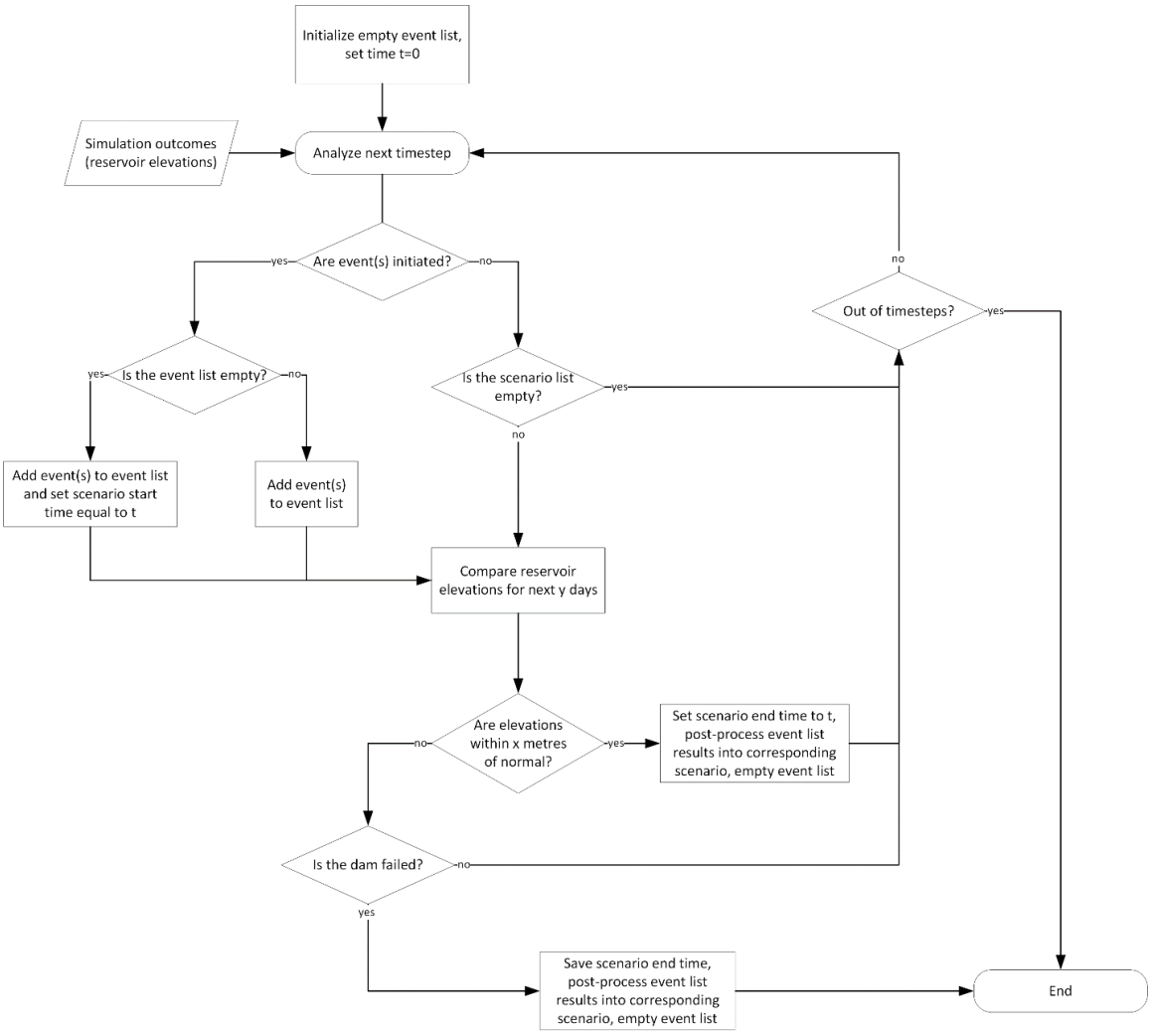


Figure 3-22: Event dependency algorithm (King and Simonovic, 2020)

This process, when applied to Outcome 1 in Figure 3-21 saves results for Scenario A, and Scenario BC. For Outcome 2, it saves results for Scenario A only. This process could also be useful to analyze outcomes from fully stochastic simulation models, extracting more information than a singular probability of failure for the system being analyzed (though this is not examined in this research).

Once all iterations for a given scenario are analyzed, the scenario results are saved and the seed number is added to the completed seeds list. Then, a new scenario is chosen from the seeds to run list and executed. Simulation of the complete list of scenarios is a significant computational task, depending on the size of the scenario list and complexity of the simulation model. Linking of each individual component in the database to the corresponding system dynamics model component is required prior to the start of the simulation. Synthetic inflow sequences are also required for the simulation, to provide more variable hydrological conditions than can typically be observed in the historical record (see Section 3.5.2). Depending on the size of the problem, it may be beneficial to use cluster computing to simulate multiple scenarios (seeds) in parallel, since the scenarios are completely independent and do not communicate between one another. More information about the cluster computing application is provided in Appendix F.

In using the Deterministic Monte Carlo approach, it is important to consider that the goal of the exercise is to analyze all scenarios (predefined combinations of operating states) as completely as possible given computational time constraints, to determine the criticality of these combinations. There should be enough data on each scenario to estimate the range of expected system performance and calculate the criticality parameters: conditional failure frequencies, failure inflow thresholds and conditional reservoir level exceedance frequencies. To ensure there is enough data collected for each scenario, it may be necessary to limit the time between events to ensure their collective impacts can be assessed. This limit may be determined as a function of the impact lengths for a given iteration (for example, by taking the sum of impact lengths). Whether the event initiation time limit should be greater than or less than the sum of the impact lengths requires experimenting with scenarios to determine how long the system typically takes to return to normal operation. For “flashy” reservoirs with relatively limited storage compared to inflows, the recovery time following a return to normal operations may be quite short – days or even hours. For reservoirs with large storage in comparison to inflows, this recovery time may be significantly longer. The recovery time may also be less than the sum of impact lengths, due to inflows that are less than the total capacity of the available flow conveyance facilities. The recovery time should influence the modellers decision regarding the appropriate time limit for event initiation. If the time limit for event initiation is too long,

there may be two or more sub-scenarios within each scenario, and not enough data relating to the collective impact of the combination of events.

3.4.4 Computational Considerations

Another important topic relevant to the simulation framework is the computational considerations. Computational efficiency is a major factor in this research, since a large number of scenarios are being analyzed, each for many iterations. The computing time for a single year and the number of scenarios to be analyzed governs the computational effort that will be required to execute the Deterministic Monte Carlo approach. As the modelled system complexity increases, so does the length of time to run a simulation. In addition, the number of scenarios is exponentially proportional to the number of component operating state-causal factor combinations. As such, a trade-off becomes evident between the level of detail to which the system is modelled and the amount of computational time the simulations will take to execute. This is one potential limitation of the approach. Each modeller may evaluate the trade-off differently, and this could result in two modellers creating different versions of the same system. Ultimately, this issue remains unavoidable within current computational abilities. That said, cluster computing is becoming more widely available and can be utilized to improve the simulation throughput. In the case study, Compute Canada cluster computing resources are leveraged to evaluate a large number of scenarios in parallel. This is made possible by the fact that each scenario can run independently of other scenarios, so a large number of cores may be used independently to run different scenarios at the same time. Despite the advantages of using cluster computing, an extremely large number of scenarios may still take a significant amount of time to evaluate, and output data storage is another potential factor that limits the level of complexity and number of scenarios that can be efficiently analyzed. The trade-off between complexity and computational effort remains. While the issue is not investigated further in this work, there are some potential directions for future research that may help to reduce the impacts of this limitation.

3.5 Simulation Model Input Data

There is a significant amount of data required to execute the simulation model. This includes site-specific physical relationships, synthetic inflows and baseline operations data for comparison with simulation outcomes to assess whether events are influenced by preceding events within the simulation.

3.5.1 Physical Relationships

The physical relationships for the system of interest are a required input of the simulation model. Stage-storage relationships relate the amount of water in the reservoir to the reservoir elevation which is used in various calculations. These relationships may be developed using bathymetry or pre-flooding lidar surveys and are typically readily available for existing dam systems. The relationship may be in the form of a table or a graph. Curve-fitting can be used to avoid interpolation calculations by creating a function that is representative of the stage-storage curve for all relevant reservoir elevations (the minimum to absolute maximum elevation that could be observed in simulation). Piecewise functions may be required for certain systems, to better capture the relationship over specific reservoir elevation bands.

Stage-discharge relationships relate the reservoir elevation to water spilling over free overflow weirs and dam structures and are another required input to the model. These can be developed using simple free-crest weir equations, and may also be readily available for the system being modelled. Again, converting the relationship to a function using curve-fitting may be desirable to avoid interpolations within the simulation (for efficiency).

Another key input is the relationship between gate position, reservoir elevation and flow, which is known as the gate rating curve. For most dam systems, these are developed in the form of either a table or curve. Two-dimensional interpolations can be used within the simulation to calculate the flow value based on inputs of reservoir level and gate position.

3.5.2 Synthetic Inflow Generation

For many dam systems, the historical record of inflows may only be as old as the dam itself and may not be a good indicator of potential variability in inflow conditions. Basing the outcomes of a dam safety assessment on the historically observed flows alone would significantly limit the analysis. As such, tools that can be applied for generation of synthetic inflows are described in this research. First, a stochastic weather generator (KnnCAD) is used to generate synthetic daily climate data, and next, a hydrologic model (Raven) is used to convert the daily climate data into reservoir inflows.

The KnnCAD weather generator is a non-parametric tool for stochastic, multi-site, multi-variable generation of climate data that was first introduced by Sharif and Burn (2006) and later modified by Prodanovic and Simonovic (2008), Eum and Simonovic (2012) and King et al. (2014, 2015). The weather generator is based on the K-Nearest Neighbour (KNN) resampling technique which reshuffles and perturbs the historical climate data to generate a longer time series with increased variability that is statistically similar to the historical record. The approach allows for easy multi-site climate data generation that preserves the spatial correlation between sites without assuming relationships between weather variables. In addition, no assumptions about the probability distributions of variables are required. King et al (2014, 2015) modified the KNN approach using a block resampling technique which was found to significantly improve temporal correlations in the resulting temperatures. These temporal correlations are extremely important in climates such as Canada where snow accumulation and melt contribute significantly to flood events. The KnnCAD weather generator was chosen for this research due to the demonstrated ability to generate statistically similar climate datasets. The most current version of the model is available on the GitHub repository FIDS-UWO/Climate and a technical manual containing the model equations, scripts and step-by-step instructions was developed by Mandal et al. (2017). A user interface makes application of this model quite straightforward. Historical time-series of climate variables are uploaded to the user interface for each site of interest. Perturbation parameters are selected – these parameters dictate the proportion of "randomness" that is applied to the climate variables. The model is then run for a user-selected number of "blocks" which are the length of the historical input data.

Following generation of climate data, hydrological modelling is required to convert the daily climate data into reservoir inflows for the site of interest. There are a number of approaches that can be used; however Raven Hydrological Modelling Framework is the platform used by BC Hydro and as such was selected for this project. Raven is a highly flexible modelling framework that allows the user to select the specific modelling code or approach to be utilized. In this research, the application is a mountain watershed, and as such the UBC Watershed Model approach is used (Quick and Pipes 1977; Micovic and Quick 1999). The UBC Watershed model utilizes daily maximum and minimum temperatures as well as precipitation to forecast snow accumulation and melt, along with soil moisture, groundwater transfer and evapotranspiration. Basin area-elevation characteristics are direct inputs to the model, which has the ability to include several elevation bands or zones that allows more realistic modelling of mountain runoff (Quick and Pipes 1977). Calibration of the model involves experimentation with specific parameters relating to various physical aspects of the system in order to find the parameter set that most closely correlates the outputs with the runoff calibration period (Quick and Pipes 1977). Combinations of different weather stations may be experimented with to find the set that provides the best calibration. Once the model is calibrated, the synthetic daily climate data from the stochastic weather generator can be used directly as an input to the calibrated model. Simulation of runoff is done in water-years which begin in October and end in September, to allow for proper continuity of snowmelt modelling. The resulting dataset is a long, synthetic inflow time series with higher variability than the historically observed data. This synthetic inflow set can be used directly in the simulation model.

3.5.3 Baseline Operations Data

Once the simulation model has been completed and synthetic inflows are developed, it is necessary to develop baseline operations data. The baseline operations data is the normal reservoir elevations, which are used in the final step of the simulation to analyze whether the events simulated within an iteration are independent of one another (and whether the iteration is “complete”). This is described in detail in Section 3.4.3.

The baseline reservoir elevations can be calculated by running a single continuous simulation of the same length as the synthetic inflows. During the simulation, no

disturbances are implemented and the system is operated normally. The resulting reservoir levels are recorded and saved to be utilized in the scenario iteration analysis following each simulation.

3.6 Scenario Outcome Assessment

Following completion of the scenario simulations, there is an extremely large amount of data from which useful information needs to be extracted. In this research, criticality parameters are used to provide information about the severity of a scenario. The criticality parameters assessed are the conditional failure frequency, conditional frequency of exceeding key reservoir levels and failure inflow thresholds for each scenario. Ranking and filtering scenario subsets (of N affected components) can give insights into system vulnerabilities and key components affecting dam safety.

Accessing and analyzing individual scenario results may also be useful. Dynamic performance measures are used in this research to better understand the dynamic system response to a given scenario. These performance measures may differ depending on the system of interest and can change from application to application. In this work, reservoir elevations over critical levels, flow conveyance capacity, and uncontrolled releases are selected and described in the following sections.

3.6.1 Criticality Parameters

The simulation environment presented in this research explores a random subset of the potential outcomes relating to a given scenario. Each scenario is simulated 2000 times, providing increased coverage of the possibility space (See Figure 3-2). This allows for an estimation of the criticality associated with a given scenario. Criticality parameters have been selected to provide useful insights about the range of outcomes simulated for each scenario. These include: the conditional failure frequency, the failure inflow thresholds, and the conditional reservoir level exceedance frequency (conditional on the scenario occurring).

As a direct result of the simulation, it is possible to determine the conditional probability of failure for a scenario from its complete iterations (all of the operating states for that particular scenario have both occurred and affected one another). This is easily done by determining the number of complete iterations, calculating the number of dam breaches occurring within the complete iterations, and determining the total proportion of failures, as follows.

$$CFF\ (\%) = \frac{N_B}{It} \times 100 \tag{3.16}$$

where N_B is equal to the number of breaches observed in all complete iterations, and It is equal to the number of complete iterations for the scenario being analyzed. Again, complete iterations are where all operating states for a given scenario have both occurred and affected one another – so iterations with multiple sub-scenarios or dam breaches occurring prior to all events being initiated are excluded from the calculation.

Another useful outcome from the simulation model is the inflow thresholds, above which failure occurs for a given scenario. In this research, the inflow thresholds are computed by looking at the 5-day average daily inflow preceding a dam breach, as well as the 5-day maximum daily inflow preceding a dam breach, taking the minimum across all simulations, as follows.

$$IT_{av} = \min\left(\text{avg}\ [I_1, I_2, I_3, I_4, I_5]_{if}\right) \tag{3.17}$$

$$IT_{max} = \min\left(\max\ [I_1, I_2, I_3, I_4, I_5]_{itf}\ \right) \tag{3.18}$$

where I_n represents the inflow on the nth day preceeding the dam breach, $n = 1\ldots5$, and s represents the simulation iteration, $itf = 1 \ldots ITF$ and ITF is equal to the number of iterations that were completely simulated for a certain scenario and where the dam failed. The minimum is taken of all maximum or average 5-day inflows preceding failure, for the complete iterations within which a failure occurred. It is also possible to consider volume of inflows in the days preceding a dam failure, however that was not explored in this research.

Another useful criticality parameter is the time it takes, following the start of the scenario, for the system to reach a failed state. This is simply calculated as the mean time to failure.

Finally, regardless of whether a dam failure occurs, there may be adverse impacts relating to exceedances above particular reservoir elevations. The likelihood of exceeding a key reservoir elevation is another easily calculated outcome from the model. The maximum reservoir elevation for each run of a given scenario can be computed and used to compute the proportion of runs where elevations exceed the reservoir level of interest, as follows.

$$CREF(\%) = \frac{I_{RSE>CE}}{It} \times 100 \tag{3.19}$$

where $I_{RSE>CE}$ represents the number of complete iterations where the reservoir elevation exceeded the critical level, CE, and It represents the total number of complete iterations. This is the conditional probability of exceedance for that reservoir elevation and scenario.

The scenarios can then be sorted based on their criticality parameters to illuminate the most troublesome operating conditions within which the system may be operating. Grouping the list of scenarios into a smaller list is possible by combining scenarios that contain the same operating states with different causal factors. This can help reduce the list size while providing extra simulation-years with which to estimate the failure frequency. If there is sufficient information to estimate the frequencies of each operating state in the model, it may be possible to compute the frequency of failure for the system using simple probability theory, as can be illustrated using a simple example.

Conditional overtopping failure frequencies for an example scenario are shown in Table 3-2 (given the scenario has occurred). When combined with the estimated frequency of occurrence of the events, an overall estimate for the frequency of overtopping failure for the system can be made using basic concepts from probability theory. A simple example is used to demonstrate this. Consider a system with components A, B, and C, which are each functional or failed. If each of the three components has lower and upper bound failure rate estimates ranging from 0.1% to 1%, an overall probabilistic assessment of the system can be made using the conditional overtopping failure frequencies generated through

Deterministic Monte Carlo simulation, as shown in Table 3-2. The conditional failure frequencies are assumed for the sake of the example.

Table 3-2: Probabilistic risk assessment using example simulation

Scenario	Conditional frequency of system failure given scenario occurs (%)	Lower bound frequency of component failure: A=0.1% B=0.1% C=0.1%	Lower bound frequency of component failure: A=1% B=1% C=1%
A	1	$9.98*10^{-4}$	$9.80E*10^{-3}$
B	1	$9.98*10^{-4}$	$9.80*10^{-3}$
C	1	$9.98*10^{-4}$	$9.80*10^{-3}$
AB	5	$5.00*10^{-6}$	$4.95*10^{-4}$
AC	5	$5.00E*10^{-6}$	$4.95*10^{-4}$
BC	5	$5.00E*10^{-6}$	$4.95*10^{-4}$
ABC	20	$2.00E*10^{-8}$	$2.00*10^{-5}$
Total probability of flow control failure		$3.01*10^{-3}$	$3.09*10^{-2}$

In Table 3, the conditional frequency of dam overtopping failure for each scenario is multiplied by the probabilities of the system states , as follows: $P(A) = P(A) * P(\bar{B}) * P(\bar{C}) * P(f)$, where $\bar{B} = 1 - B$, and the solid line over the component indicates it is not failed, and $P(f)$ represents the conditional probability of overtopping failure for the system given the scenario has occurred. In the table, the lower and upper bound estimates are calculated to illustrate the sensitivity of the results to the assumed component failure probabilities. This is particularly advantageous where failure rate data is limited and uncertain. The Deterministic Monte Carlo approach does not require complete re-simulation if the sensitivity of the results to the assumed probabilities is to be analyzed. The sensitivity of the overall probability of failure for the system can be easily calculated by simply modifying the assumed component failure rates and updating the equation. In contrast, a fully stochastic simulation approach would require re-simulation to analyze the sensitivity of results to assumed failure rates, since the probabilities are embedded within the stochastic simulation model.

In the absence of reliable information relating to the failure of various components, the overall failure rates were not explored further in this research. This topic remains an important area for future work.

3.6.2 Performance measures

In terms of overall assessment of the system performance, it is useful to define dynamic safety performance measures that can be plotted over time from the scenario outputs. These performance measures show how the system changes over time and, where possible, the recovery from the disturbance. Different performance measures may be selected for different systems of interest. In selecting these performance measures, it is important to consider the functions which a dam is meant to carry out and how the system may reach a less desirable state.

Dam systems act to store and convey water for beneficial purposes such as hydropower, water supply and flood control. The dam acts to retain water and its flow-conveyance features (eg. spillways, turbines, low level outlets and valves) are controlled by dam operators to pass water and maintain reservoir levels within safe limits. Loss of control of the reservoir can occur as a result of natural disturbances such as earthquakes, landslides, debris, etc., as well as a number of internal factors including operational failures, inflow forecasting errors, site access and staffing problems as well as systemic problems like failing to maintain and upgrade infrastructure. Loss of functionality of flow-conveyance features of the system can directly lead to loss of reservoir control, potentially causing overtopping and failure. Issues with dam design or external disturbances can also affect the dam itself resulting in the inability to retain water which could potentially lead to dam collapse and catastrophic flooding. In considering the functions a dam is meant to perform, it becomes clear that two key performance measures relate to flow retention and flow conveyance. Flow conveyance capacity and uncontrolled flow releases are chosen to represent flow conveyance and retention, and reservoir elevations exceeding critical safety levels is also selected. These performance indicators and their values over time can be calculated directly from simulation outputs. The result is a numerical indicator showing how the dams condition changes with time for a given operating scenario.

It is ultimately up to the experts and asset owners to determine an appropriate amount of detail for the simulation model and select a particular set of performance measures of interest for a specific system. The following sections describe the performance measures selected for this research, but others may be added depending on the dam of interest. A final section describes aggregation of scenario outcomes to reach general conclusions about the performance of the dam.

3.6.2.1 Conveyance Capacity

Conveyance capacity represents the ability of flow-conveyance infrastructure such as turbines and spillways to pass water through the system. This is an important indicator of system safety because a loss in conveyance results in a lowered ability to manage inflows safely. Conveyance capacity is equal to the available total discharge capacity as a function of time (Equation 22):

$$CC(t) = \sum_{c=0}^{C} FC(c,t) \tag{3.20}$$

Where $CC(t)$ is the discharge capacity of the system at full pool for time t, and $FC(c,t)$ is the discharge capacity of flow-conveyance component c $(c = 1 \dots C)$ at full pool for time t. If all conveyance facilities are performing, the maximum performance value is thus equal to the maximum discharge at full pool, including free overflow facilities (and the minimum performance is 0 m^3/s).

3.6.2.2 Total Uncontrolled Release

Another key indicator of dam system safety is the ability of the system to retain water where it is meant to do so. Failure to retain water results in an uncontrolled release of flow, which may be through the dam itself (dam breach), or through a failed penstock, spillway gate, or turbine head cover. Uncontrolled release also includes any water passing over the free-crest spillway and dam, which represents flow that is no longer under the control of the operators. Total uncontrolled release (UR) is calculated at each time step and is shown in Equation 22:

$$UR(t) = QDB(t) + QPL(t) + QOF(t) + QHC(t) + QGC(t) \tag{3.21}$$

Where QDB is the dam breach flow, QPL is the penstock leakage, QOF is the flow passing through the overflow weir or over the dam, QHC is any water escaping through the head cover of the turbine, and QGC is any water passing through a failed spillway gate. The individual uncontrolled release variables are useful on their own as well, and can be investigated for a particular scenario and iteration directly from the model output. Combining these into a single variable, UR, provides some useful indication about the performance and helps reduce the size of the simulation output files, but when there are multiple sources of uncontrolled release it may become more difficult to analyze what the sources are. This is a minor limitation that can be overcome by saving these flows separately if additional data storage capacity is available.

3.6.2.3 Reservoir elevations exceeding critical safety levels

Perhaps the most important indicator of dam system safety is the reservoir elevation itself. Dam systems typically have reservoir operating limits within which the reservoir remains, known as the normal minimum and normal maximum flows. Some excursions above the maximum level may be expected during high inflow conditions, and there may be a safety-critical reservoir levels which the reservoir should not exceed due to potential dam safety problems. For an earth dam, the elevation of the core or filter material should not be exceeded as this may result in internal erosion and could potentially progress to dam failure. For a concrete dam, there may be other factors such as structural stability being reduced above a certain reservoir level. This elevation will differ between dam systems and could be equal to the height of the dam itself. Elevations over critical safety levels can be visualized in two ways: (1) by observing the resulting reservoir level plots for each complete iteration of a scenario, where all scenario events both occurred and affected one another, and (2) through reservoir level time exceedance frequency plots. These plots can be easily derived by collecting all observations for each complete scenario iteration and determining the percentage of time that various elevations are exceeded using the following formula (USBR 2018):

$$EF_{ref\ el.} = \frac{Nobs - \left(Nex_{ref\ el.} + 1\right)}{Nobs + 1} * 100 \qquad (3.22)$$

Where $EF_{ref\ el.}$ is the exceedance frequency for a specific reservoir elevation $ref\ el.$, $Nobs$ is the total number of observed daily reservoir elevation values from the scenario's complete iterations (all events occurring and affecting one another), and $Nex_{ref\ el.}$ is the number of observations where the elevation exceeded $ref\ el.$ To generate exceedance frequency curves, a range of reservoir elevations is taken from the minimum to the maximum at a user-defined interval, and the exceedance frequency is computed for each. The reservoir levels are then plotted against the exceedance frequencies. Key critical levels (such as free overflow spillway sill elevation and the elevations of key structures) can be plotted on the exceedance frequency curves to help illustrate the severity of the scenario.

3.7 Summary

This chapter presents the methodology for the research. First, a description of the requirements of a new approach and the ability of existing tools to meet these requirements is provided. While each approach offers specific advantages, there are limitations inherent to all of the approaches used within and outside of the dams industry. This leads to the methodology development, which aims to meet as many of the requirements as possible. This research proposes using a systems approach to the problem of dam safety analysis, systematically characterizing pre-generated scenarios through simulation.

A new methodology is presented that uses Deterministic Monte Carlo simulation to analyze a wide range of potential operating conditions for a dam system. Scenarios are used as deterministic inputs to the model, and the scenario parameters are varied using Monte Carlo techniques to explore each scenario's potential outcomes. In order to generate a list of scenarios for the simulation model, an operating states database is developed which can be applied to any system, and used to document components, their operating states, causal factors, and operating state impacts. Using database outputs and component operating state sets, a combinatorial procedure applies the Cartesian Product to come up with the complete range of component operating state combinations (scenarios). The scenarios becomes the input to the Deterministic Monte Carlo simulation framework.

The simulation framework uses the pre-generated scenarios (operating states) as inputs, with Monte Carlo variation of inflows as well as operating state impact timing and

magnitude. This simulation framework has the advantage of (a) investigating a larger, more complete set of potential scenarios than is practical using traditional methods, and (b) providing a more in-depth analysis of the range of system behaviour in response to each scenario.

Simulations are performed using a system dynamics simulation model, which is capable of representing complexity, feedbacks and component interactions in a relatively straightforward way. The object-oriented modelling environment used in system dynamics clearly shows the components and the relationships between them which improves the transparency of the model and the ease with which it is built and modified. Timing considerations are also addressed in this work. An algorithm is proposed to assess whether preceding events within a simulation affected the events that occurred later.

The results from the simulation can be analyzed in a number of ways. Post-processing of individual scenarios can be performed to determine the conditional probabilities of failure and excursions above key reservoir elevations, as well as inflow thresholds for failure. Individual scenario results can be used to plot the reservoir elevations, flow conveyance capacity and uncontrolled releases over time, as well as reservoir exceedance frequency plots. The methodology proposed in this work provides a means of evaluating the full range of possible operating state combinations for the system within current computational abilities. At this time, the same outcome would not be possible using stochastic techniques because the occurrence of these combinations of events is quite rare (they have a low probability), so a prohibitively large number of simulation-years would be required to achieve the same result. The methodology in this research also evaluates scenarios dynamic outcomes, taking into account feedbacks and nonlinear behaviour which are not readily dealt with using the traditional risk assessment techniques.

Chapter 4

4 Case Study: Cheakamus Hydropower Project

The methodology presented in the previous chapter has been applied to BC Hydro's Cheakamus Hydropower Project. A complete database representation of the system is presented, and the combinatorial procedure is used to generate all combinations of component operating states (scenarios). A full detailed system dynamics model representative of the Cheakamus Project is described in King et al. (2017). Due to an extremely large number of potential scenarios for the case study, a simplified proof-of-concept example was subsequently developed, which has some of the characteristics of the Cheakamus Project. The key difference is a reduced number of system features, with the goal of reducing the number of potential scenarios to ensure simulation feasibility with the limited computational resources available. With limited computational resources on the Compute Canada systems, it was possible to simulate two complete runs through 1.11 Million scenarios, each with over 1000 different Monte-Carlo input parameters (iterations). The two runs completed were the base case and the dam safety improved case which contained modifications to operating rules and components.

The following section provides a description of the Cheakamus Hydropower Project which was the study area for this research. Next, a description of the system dynamics model development is provided for the detailed representation of this system. The following section deals with scenario generation for the detailed Cheakamus representation. Next, a description of the simplified version of the Cheakamus system is provided, due to the extremely high number of scenarios generated for the complex system representation. The simplified generation of scenario is described, as well as a detailed description of the simulation model configuration and testing. Inflow generation for the case study is presented, followed by a description of the scenario simulation process. Finally, simulation results and discussion are provided.

4.1 Cheakamus Hydropower Project Description

The Cheakamus Hydropower Project is located 30km north of Squamish, British Columbia, Canada and is operated by BC Hydro, the provincial power utility. The Cheakamus River originates approximately 25km southeast of Whistler, B.C. and has an area of 1070km^2. The headwaters start at 2500m above sea level and the river eventually discharges into the Squamish River 26km downstream of the dam at El. 30m (above sea level). Cheakamus Dam impounds Daisy Lake and has a drainage area of 780km^2, receiving about 75% of the Cheakamus river inflow (BC Hydro 2005). The average reservoir inflow is around 50m^3/s (BC Hydro 2005).

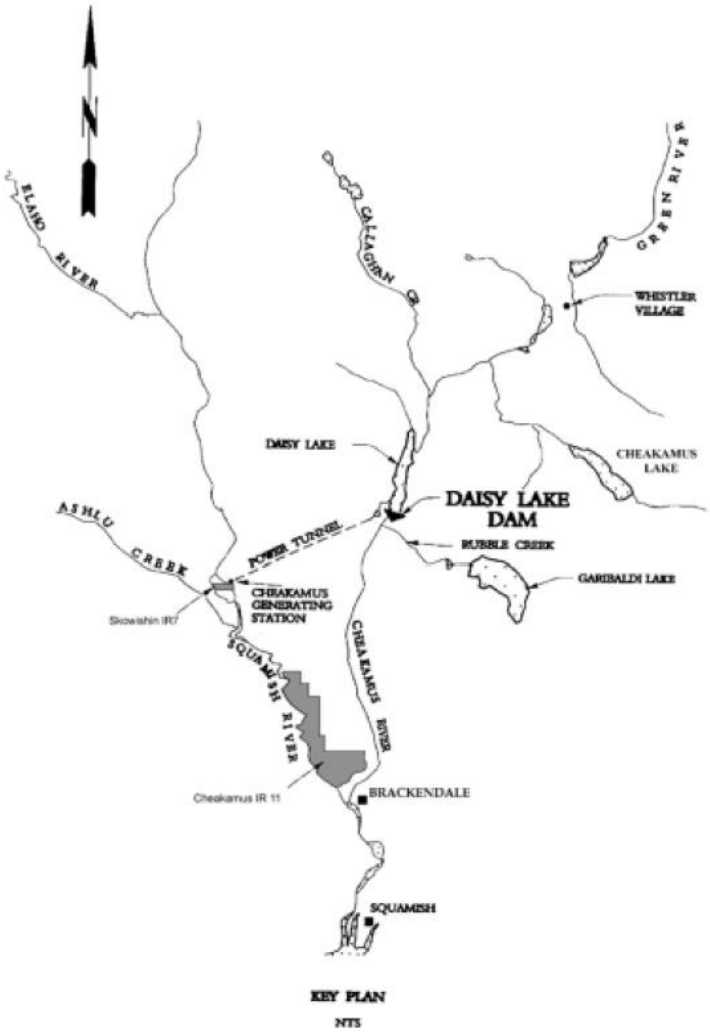


Figure 4-1: Cheakamus Hydropower Project area map (BC Hydro, 2005)

Figure 4-1 contains a map of the region with the locations of the dam and powerhouse shown. Daisy lake has a live storage capability of 55 million m^3 and a typical operating range of El. 364.90m to El. 377.25m (BC Hydro 2005). The stage-storage curve for Daisy Lake is provided in Appendix A.

The Cheakamus Main Dam consists of an Earthfill Dam, a Concrete Main Dam gravity structure, and a concrete gravity overflow Wing Dam. Daisy Lake is also impounded Saddle Dam No 1. An overflow channel, along with the Wing Dam and Saddle Dam No. 1, provide free overflow discharge capability for the system. A power canal leads to the power intake structure at Shadow Lake which is impounded by the Shadow Lake Saddle Dam. Water for power is drawn through a canal beneath Highway 99 and into an 11km tunnel through Cloudburst Mountain. At the end of the tunnel, two penstocks carry the water to a powerhouse that discharges into the Squamish River upstream of its confluence with the Cheakamus River. The maximum power discharge is $65m^3/s$ which can generate up to 157MW of power through two vertical Francis units. Flood flows are discharged into the Cheakamus River at the Concrete Main Dam, which contains two Spillway Operating Gates (SPOGs) with a combined discharge capacity of $1590m^3/s$ at the maximum normal reservoir level (MNRL) of El. 378.26m. A low level outlet sluice (LLO) with a discharge capacity of $196m^3/s$ at MNRL and five free overflow spillway ports are also located at the concrete dam. There is an additional low level outlet controlled by a Hollow Cone Valve which is considered to be out of service. Details regarding the relationship between elevation and discharge for fully open gates are provided in Appendix A. The project schematic is shown in Figure 4-2. An overall site plan showing the locations of the dams can be found in Figure 4-3.

The province of BC underwent a water use planning process for Cheakamus Dam that prescribed minimum discharges downstream of the dam, flow ramping rates (rates of discharge increase) and operating levels to be adhered to (if possible) by the system operators. The minimum discharge information is shown in Appendix A (BC Hydro 2005). The historical daily inflows are also shown in Appendix A.

CHEAKAMUS PROJECT

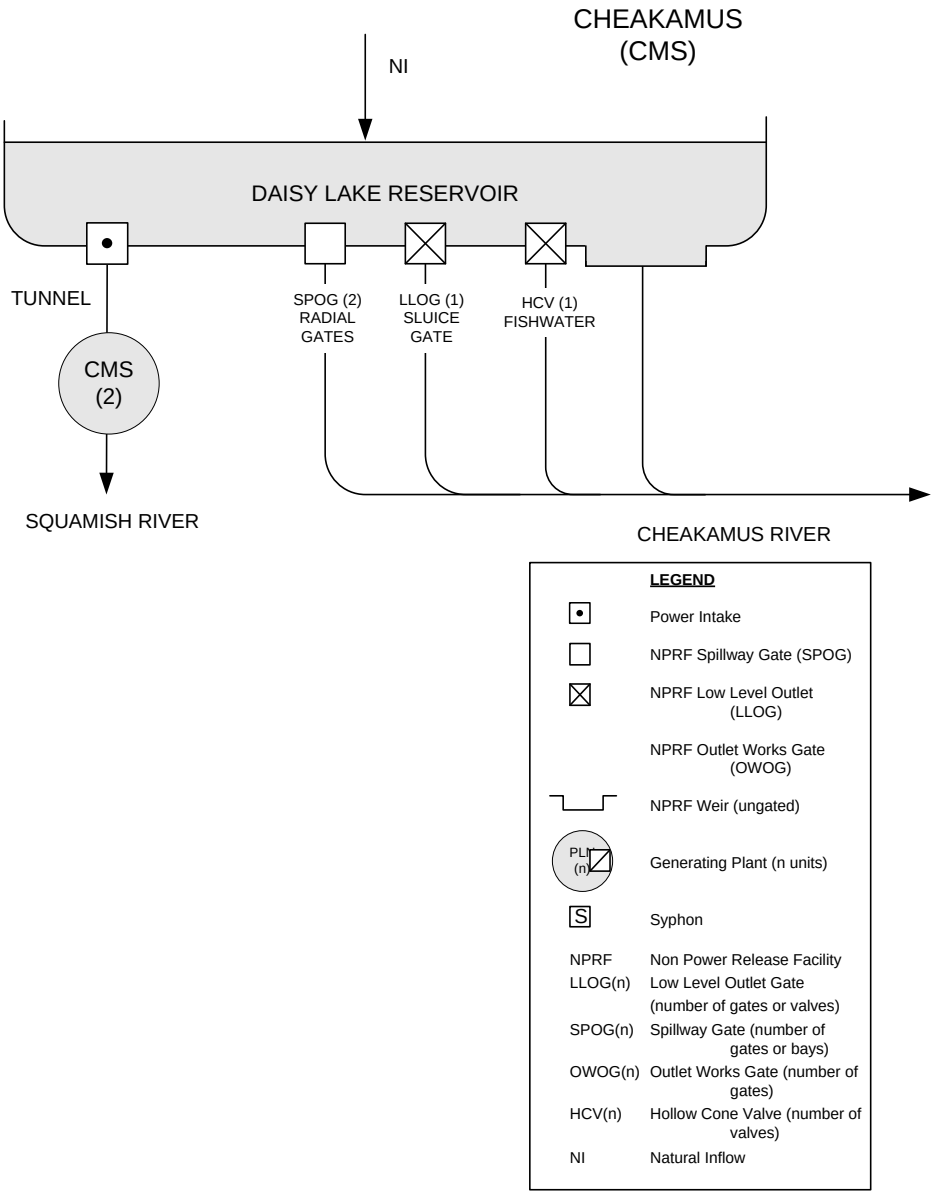


Figure 4-2: Cheakamus Hydropower Project system schematic (BC Hydro, 2005)

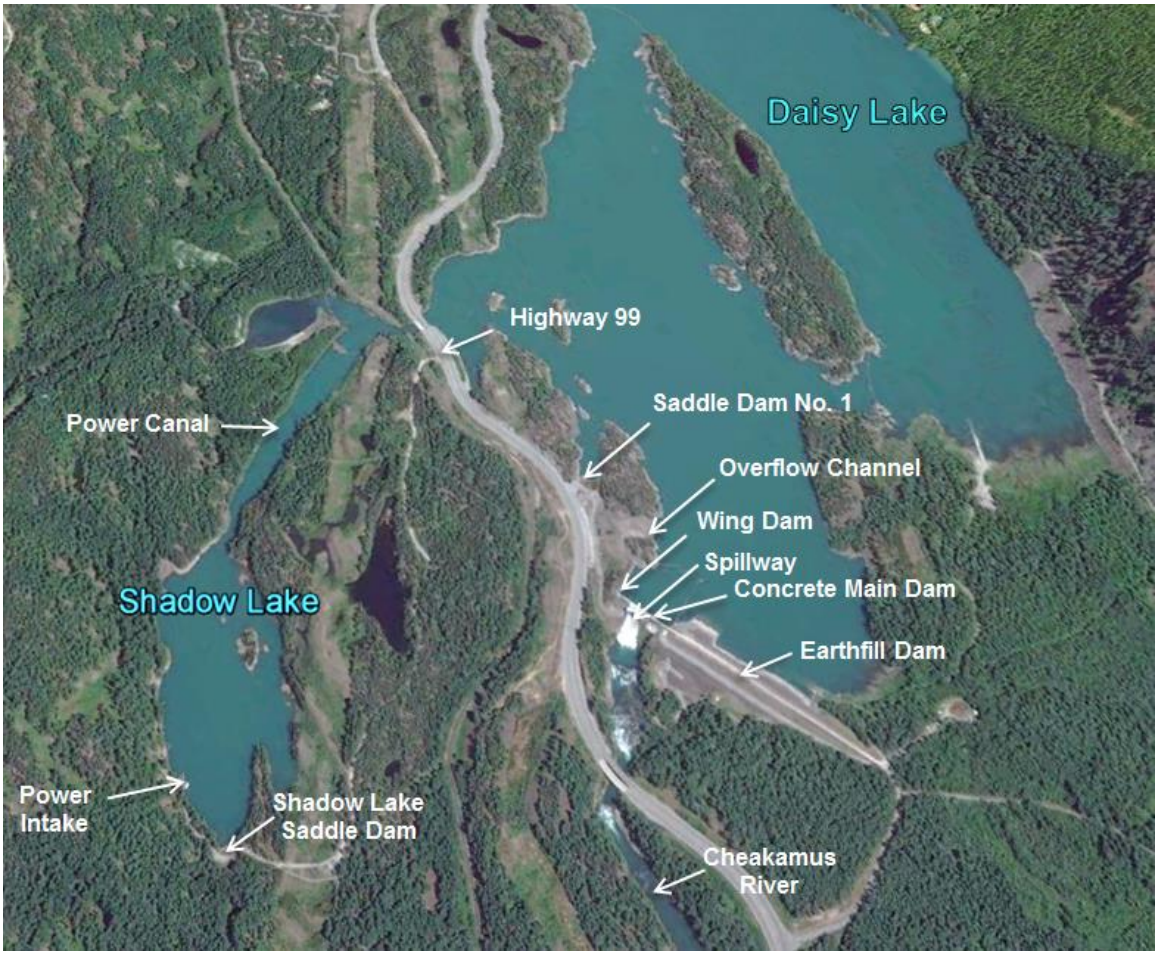


Figure 4-3: Cheakamus dam site overview

It is also useful to understand the control system infrastructure in place for the Cheakamus System. A hierarchical control system structure is shown in Figure 4-4. This schematic shows the regulatory and organizational controllers at the top, moving down towards the control infrastructure itself. The exchange, transfer and movement of information throughout the system is shown in detail in this figure.

The Cheakamus System structure and data were used with the framework described in Section 3.2 to populate the component operating states database for the Cheakamus System and generate an extensive list of potential operating scenarios (Section 3.3). This process is described in the following section.

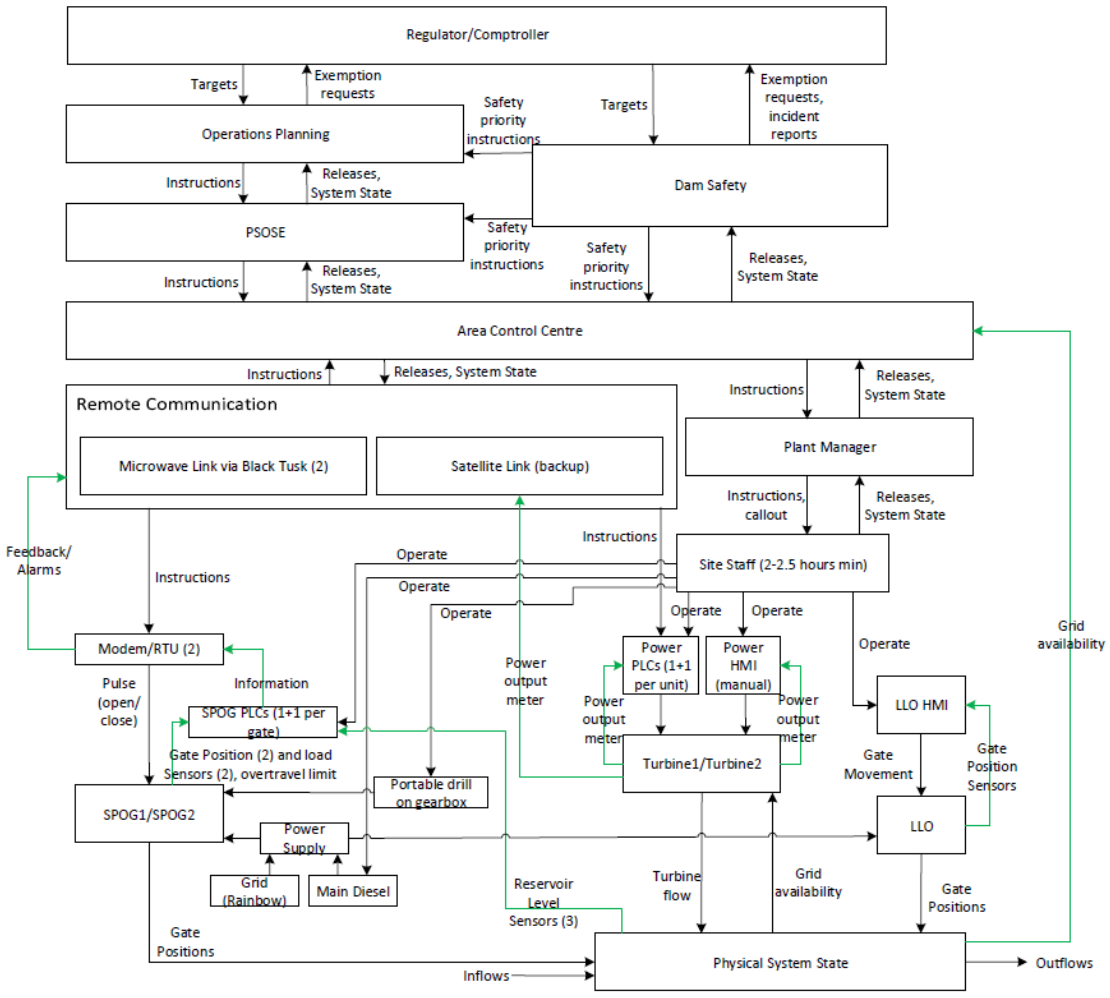


Figure 4-4: Hierarchical control system structure of Cheakamus Project

4.2 Cheakamus Database Population and Scenario Development

To generate scenarios for the Cheakamus System, an in-depth understanding of the system and its interactions is required. Information should be collected about the system structure, components and connections. This will help in identifying the components and their potential operating states within the database. In this research, STPA is used to improve the understanding of the system for database population. The process also helps inform the development of the system structure within the simulation model. Following STPA, the

database is populated and the database outcomes are used to generate the complete list of scenarios.

4.2.1 Systems Theoretic Process Analysis for Cheakamus System

In order to help facilitate the development of the operating states database, the STPA procedure of Leveson (2011) was applied to a high-level version of the detailed Cheakamus system. STPA is a systematic approach to evaluating potential control actions that can lead to hazards for a system. The control actions can then, if possible, become operating states within the database or can be programmed into the simulation model.

The goal of the STPA analysis was a high level review of potentially hazardous conditions at the dam site, to help guide the development of the model and operating states database. The control system structure developed for this process is shown in Figure 4-4 and Appendix B contains the complete analysis that was done (though this would likely change and become much more comprehensive with expert input from BC Hydro).

Prior to initiating an STPA analysis, the high-level system hazards must be defined. The hazards selected for Cheakamus Dam are as follows:

- H1: High flows released into Cheakamus River and/or Squamish River (flood)
- H2: Flow releases to Cheakamus River stopped (fish kill)
- H3: Equipment damaged (economic/safety impact)
- H4: Loss of power production (economic impact)

Next, a set of high-level system safety constraints (requirements) are defined, as follows:

- SH1: Flows released into Cheakamus and/or Squamish must not exceed a level that causes damage downstream
- SH2: Flow must always be released to Cheakamus River
- SH3: Equipment must not become damaged

Following the definition of these hazards and safety constraints or requirements, a detailed hierarchical control system structure can be developed, as shown in Figure 4-4. Then, the process can begin, with the first step being to identify unsafe control actions. Unsafe control actions are defined for each control feature of the system, which in this case

includes both gates, both turbines and the low-level outlet. A table is used with four columns that can be used to guide the assessment: providing causes hazard, not providing causes hazard, wrong timing or order causes hazard, and stopping too soon or applying too long causes hazard.

Once the unsafe control actions (UCA's) are defined, the next step in the process involves looking at each UCA individually and considering how the issue may arise (the causes). Finally, additional factors can be listed.

Looking at the detailed description of UCA's and their causes (Step 2 in Appendix B) provides some interesting insights regarding the degree to which the approach presented in this work is able to fully cover the range of potential operating states. Focusing on the scenarios that involve flooding (H1), there are some instances where human factors may lead to more complicated scenarios than the automated procedure developed in this work is able to generate. Some scenarios may require additional effort for simulation due to the complex nature of human decision making. UCA1/UCA10 from the STPA analysis is presented below to illustrate this.

UCA1/UCA10: SPOG Open command not provided when water level high, inflow high or both [H1, H3]

Case 1: Water level high, inflow low, open command not provided

-Controllers (OP, PSOSE, ACC, DS) unaware of reservoir level due to gauge failure, sensor failures or communication delays

-High tides at Squamish mean there are flooding impacts when additional flows are released from the CMS system. Controllers (OP, PSOSE, ACC, DC) make a decision to hold water back, allowing the reservoir to rise to an unsafe state even though the inflow is relatively low.

Case 2: Water level high, inflow high, open command not provided

-Controllers (OP, PSOSE, ACC) believe they can return the reservoir to a safe level using the powerhouse and/or LLO and/or other SPOG due to inflow forecast errors

-Controllers (OP, PSOSE, ACC, DS) unaware of reservoir level due to gauge failure, sensor failures or communication delays

-High tides at Squamish mean there are flooding impacts when additional flows are released from the CMS system. Controllers (OP, PSOSE, ACC, DC) make a decision to hold water back, allowing the reservoir to rise to unsafe levels

-Controllers do not follow procedure (human error due to fatigue or shift change at PSOSE/FVO)

Case 3: Water level low, inflow high, open command not provided

-Controllers (OP, PSOSE, ACC) believe they can keep the reservoir at a safe level without opening the gate, due to inflow forecast errors or process errors

-Gate(s) out of service for maintenance purposes and therefore cannot be opened.

-Controller thinks gate open (sensor failure, communication delay)

In this example, there is a potential scenario where high tides at Squamish (downstream of Cheakamus) lead the operator to hold back water when a high-inflow event is occurring. This scenario would be difficult to analyze within the proposed Deterministic Monte Carlo model, due to the major factor being human decision making. It would require additional simulation effort to fully capture this potential scenario. Process errors or controllers not following procedure are difficult to simulate since there are so many different ways in which the decision-making can unfold. Some of the causes of the UCA shown above (for example gates being out of service, sensor failures, communication delays, inflow forecast errors, etc.) are both easily incorporated into the operating states database, and easily simulated using the system dynamics model.

STPA is quite useful as a preliminary assessment tool that can be used to inform the development of the operating states database and simulation model. The results of the assessment can also significantly help with identifying and some non-failure related operating constraints that have the potential to lead to a hazard – the operating states database and simulation model can then be developed to ensure simulation of these non-failure related events is possible. Some of the scenarios uncovered through this systematic assessment approach would be difficult to quantify using the automated simulation approach described in this work and may not fit well into the database structure. However, they may be able to be analyzed through a more case-specific simulation experiment. Performing an STPA is helpful to improve the understanding of the system and ensure operators are aware of all potential causes of failure for the system in order to manage risks and avoid catastrophic impacts of dam failure.

It is important to note that the STPA analysis in Appendix B is provided for illustrative purposes only. It is in no way representative of a complete assessment for the real Cheakamus System, and was not performed by BC Hydro personnel.

4.2.2 Database Population and Scenario Generation

The component operating states database was populated based on the components in the Cheakamus Hydropower Project and the information gathered through the STPA process. The components tree showing the system configuration is provided in Figure 4-5. Each component in the leftmost column is at the Reservoir Level. Each drop-down to the right of this consists of the Component Level features of the system. Each of the components contains a minimum of two operating states (normal and adverse) and each operating state has a minimum of one causal factor. Each combination of operating state and causal factor is recorded as a separate operating state. The complete database extract table for the complex system is presented in Appendix C.

The information in the database is used to come up with a unique identifier for each object in the system as well as it’s causal factors. For Reservoir Level components, the

ReservoirLevelID is used as the component identifier, n, and a number m_n, $m_n \in (1 \ldots M_n)$ is assigned to each operating state/causal factor combination for component n. The operating state identifier takes the form n_m_n, which is used to group the operating states into sets for each component that are used in the calculation of the Cartesian product. For objects at the component level, the ReservoirLevelID and the ComponentLevelID are combined into a three to four-digit number which is used as the component identifier, n, since there may be multiple items at the Component Level for a single Reservoir Level item. Operating state/causal factor combinations are similarly assigned a number m_n. The identifiers can be seen in the database extract table in Appendix C Once the identifiers are assigned, they are grouped into sets and Python's *itertools product* function is used to compute the Cartesian Product, which results in a list of all possible combinations of operating state identifiers for each component.

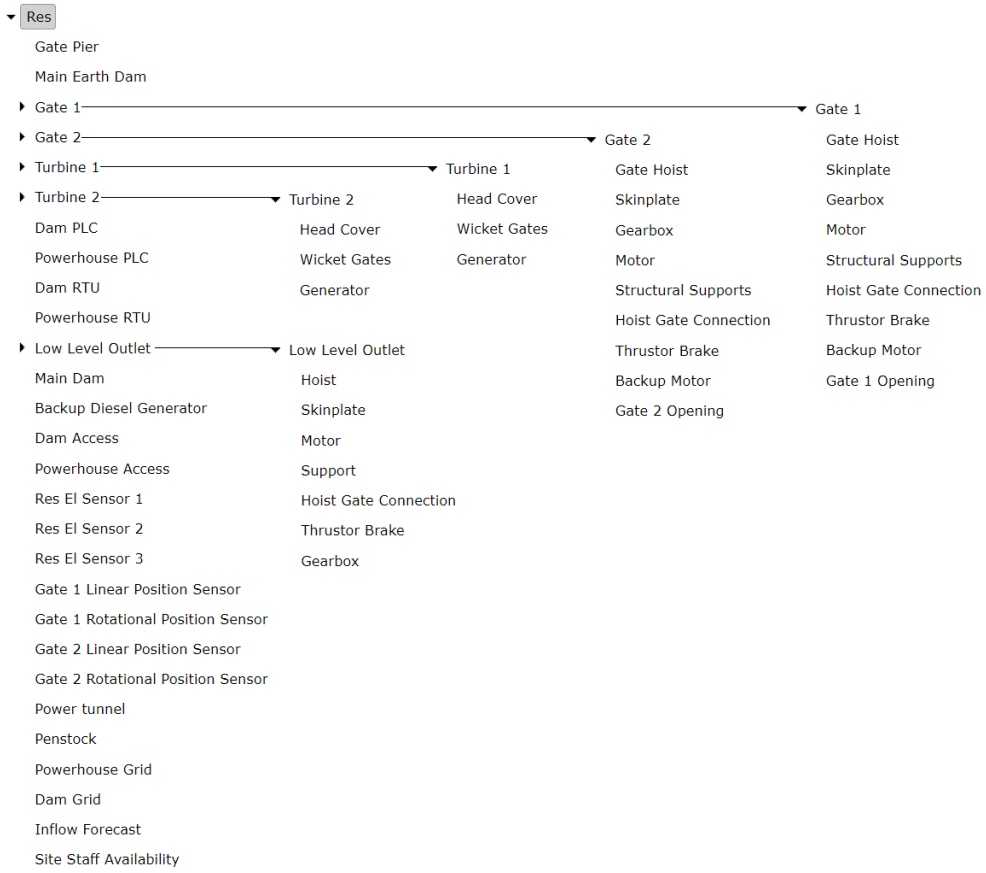


Figure 4-5: Components tree for the Cheakamus System

Table 4-1 contains a list of each component in the system and its object identifier, *n*, which is a two-digit number for Reservoir Level items and a three to four-digit number for Component Level items. The name of each component is shown as well as the total number of operating states in each operating state set, M_n. Multiplying together all of the numbers in the column M_n, as per Equation 3 gives the total number of possible scenarios, or the number of elements in the Cartesian product, which is equal to 1.83×10^{27}. This number can be verified by computing the Cartesian product using Python's *itertools product* function, which generates a list of the same length. Each element in the generated list contains a single operating state for all components in the system. This is an exhaustive list which includes everything from a completely functional system to a system where every component has some adverse operating state.

An additional calculation was done where causal factors leading to the same operating state were grouped as a single operating state. This would avoid redundant simulations of the same operating states with different contributing causal factors – though the model does distinguish between causal factors in terms of time-of-year restrictions. The number of scenarios with grouped causal factors for the Cheakamus system model is 1.54×10^{17}, which is significantly fewer scenarios than if each causal factor-operating state combination is considered separately. Grouping of causal factors to avoid redundancy may potentially be an effective way to reduce the number of simulations required to evaluate each scenario.

Obviously, simulation of such large number of scenarios would be computationally prohibitive given the current state of technology and the finite resources available for this research project. As such, the simulation portion of this research is focusing on a simplified abstraction of the Cheakamus System, described in the following section.

Table 4-1: Number of unique operating state and causal factor combinations for each component in the complex system

Object ID, *n*	ReservoirLevelID	ComponentLevelID	Reservoir Level Name	Component Level Name	M_n
46	46		Gate Pier		2
22	22		Main Earth Dam		1
18	18		Dam Programmable Logic Controller		3
19	19		Powerhouse Programmable Logic Controller		3
16	16		Dam Remote Terminal Unit		3
17	17		Powerhouse Remote Terminal Unit		3
21	21		Main Dam		2
27	27		Backup Diesel Generator		5
29	29		Dam Access		5
28	28		Powerhouse Access		5
30	30		Reservoir Elevation Sensor 1		7
31	31		Reservoir Elevation Sensor 2		7
47	47		Reservoir Elevation Sensor 3		7
37	37		Gate 1 Linear Position Sensor		4
39	39		Gate 2 Linear Position Sensor		4
38	38		Gate 2 Rotational Position Sensor		4
36	36		Gate 1 Rotational Position Sensor		4
41	41		Power tunnel		2
42	42		Penstock		2
44	44		Powerhouse Grid		5
43	43		Dam Grid		5
45	45		Inflow Forecast		3
48	48		Site Staff Availability		3
1326	13	26	Gate 1	Gate Hoist 1	6
1328	13	28	Gate 1	Skinplate	3
1331	13	31	Gate 1	Gearbox	3
1332	13	32	Gate 1	Motor	4
1333	13	33	Gate 1	Structural Supports	4
1334	13	34	Gate 1	Hoist Gate Connection 1	2
1343	13	43	Gate 1	Thrustor Brake	3
1355	13	55	Gate 1	Backup Motor	4
1357	13	57	Gate 1	Gate 1 Opening	2
1416	14	16	Gate 2	Gate Hoist 2	6
1418	14	18	Gate 2	Skinplate	3
1421	14	21	Gate 2	Gearbox	3
1422	14	22	Gate 2	Motor	4
1423	14	23	Gate 2	Structural Supports	4
1424	14	24	Gate 2	Hoist Gate Connection 2	2
1444	14	44	Gate 2	Thrustor Brake	3
1456	14	56	Gate 2	Backup Motor	4
1458	14	58	Gate 2	Gate 2 Opening	2
836	8	36	Turbine 1	Head Cover	2
837	8	37	Turbine 1	Wicket Gates	2
838	8	38	Turbine 1	Generator	2
1039	10	39	Turbine 2	Head Cover	2
1040	10	40	Turbine 2	Wicket Gates	2
1041	10	41	Turbine 2	Generator	2
1546	15	46	Low Level Outlet	Hoist	6
1547	15	47	Low Level Outlet	Skinplate	3
1548	15	48	Low Level Outlet	Motor	4
1549	15	49	Low Level Outlet	Support	4
1550	15	50	Low Level Outlet	Hoist Gate Connection	2
1551	15	51	Low Level Outlet	Thrustor Brake	3
1554	15	54	Low Level Outlet	Gearbox	3

4.3 Simplified System Database Population and Scenario Development

Due to the extremely large number of scenarios generated for a complex representation of the Cheakamus System, a simplified abstraction of the system was developed to facilitate scenario simulation. The goal of this simplification was to create a system that mimics the function of Cheakamus but has significantly less components and as such fewer scenarios to simulate. This simplified system provides a proof-of-concept for the methodology described in this research. For applications to similar systems to Cheakamus, it may be desirable to utilize some of the simplifications described here, such as aggregating components with similar impacts into grouped components, with the goal of reducing the occurrence of redundant scenarios. This process could potentially be guided by the use of fault tree analysis for sub-systems such as the gate equipment. Due to computational resource limitations for this research, the simplified system lacks some of the key redundancy features of the real Cheakamus System that increase its overall level of safety. As such, results for the simplified system are not considered to be representative of the real Cheakamus Project safety and performance.

In the simplified version of the system, the existing reservoir stage-storage relationship is used. The two spillway gates (SPOG1 and SPOG2) are combined into a single gate (SPOG) with a rating curve equal to the sum of the discharge columns from the individual rating curves. The Low Level Outlet sluice is omitted from the model. The two turbines are combined into a single unit capable of conveying the total flow of both units. The main communications equipment in the Cheakamus System (the PLC and the RTU) are idealized as a single component (PLCRTU). Gate components are also simplified into categories representing the impacts that occur upon failure – gate failing in place, gate failing closed, and gate collapse. The sensors for the reservoir level are combined into a single sensor and the sensors for gate position are omitted. Wicket gates are eliminated from the turbine components and a single grid is modelled instead of separate power connections to the powerhouse and the dam. In addition to the component changes, some causal factors were omitted from the analysis to reduce further the scenario list, since each operating state-causal factor combination is counted as a unique operating state.

To show how the model can be useful in assessment of safety improvements realized by potential capital upgrades and operating rules, two cases are simulated. These are a "base case" which has a significantly smaller free overflow spillway than the real system, and a "dam safety improved case" which has an identical free overflow spillway as well as improved operating rules and a slightly reduced failure frequency for certain components. These scenarios are described further in Section 4.4.3.

Using the simplified abstraction of the Cheakamus System, a new version of the database was developed. Figure 4-6 contains the components tree for the simplified system. Appendix D contains the full database details for the simplified system. Table 4-2 contains a list of each component in the simplified system including its identifier, n, the ReservoirLevelID and ComponentLevelID (from the database), component description and the number of operating states in each component's operating state set, M_n.

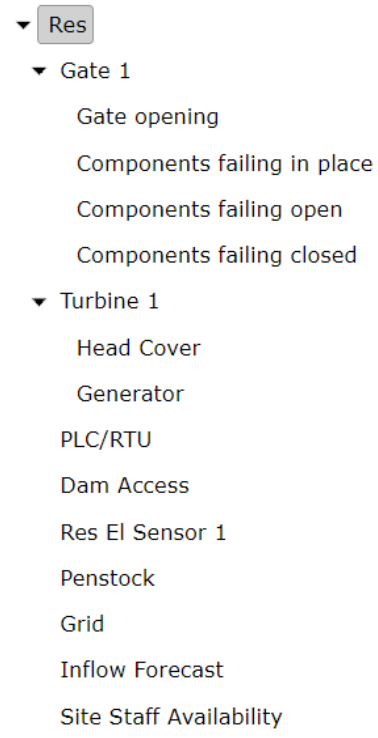


Figure 4-6: Components tree for the simplified system

Multiplying together the values of M_n as shown in Equation 3 yields the total number of possible scenarios for the system, which is equal to 5.5 x 10^5. This number can be verified

by computing the Cartesian Product using Python's *itertools product* function, which generates a list of scenarios of the same length.

Table 4-2: Number of unique operating state and causal factor combinations for each component in the simplified system

Object ID, *n*	ReservoirLevelID	ComponentLevelID	Reservoir Level Name	Component Level Name	M_n
18	18		Programmable Logic Controller / Remote Terminal Unit		4
29	29		Dam Access		4
30	30		Res El Sensor 1		5
42	42		Penstock		2
44	44		Grid		4
45	45		CMS Inflow Forecast		1
48	48		Site Staff Availability		3
836	8	36	Turbine 1	Head Cover	2
838	8	38	Turbine 1	Generator	2
1359	13	59	Gate 1	Gate opening	2
1361	13	61	Gate 1	Components failing open	3
1362	13	62	Gate 1	Components failing closed	4
1360	13	60	Gate 1	Components failing in place	3

Comparing the total number of possible scenarios for the complex representation of Cheakamus to the simple system, it is clear that with more components and a larger number of operating states and causal factors, the number of possible operating scenarios grows exponentially. There is a trade-off between the level of complexity represented and the amount of computational effort required – this requires serious consideration in model development and could result in different modellers creating different versions of the same system. It is very important to ensure any simplifications of real-world systems take into consideration component redundancies that can significantly improve scenario outcomes. Future work should explore methods for reducing the impact of this tradeoff by decreasing the computational effort required to cover larger numbers of scenarios – this may include the use of pattern recognition techniques.

Table 4-3 and Table 4-4 contain the key database parameters from a single example scenario for the simple system, for the components at the Reservoir Level and the Component Level, respectively. The example scenario contains the following identifiers (in the form Component_OperatingStateNumber):

[18_1, 44_1, 30_1, 45_1, 29_4, 42_2, 48_3, 838_1, 836_2, 1359_1, 1360_2, 1361_1, 1362_1]

Table 4-3: Database information from example scenario: Reservoir Level

Identifier	Reservoir Level Name	Operating State Name	Min	Max	Avg	Causal Factor Name	Max Date	Min Date
18_1	PLC/RTU	PLC offline	1	24	6	Voltage Fluctuation	365	0
44_1	Grid	Grid failure	0.04	7	0.16	Wind storm	365	0
30_1	Reservoir Elevation Sensor 1	Wrong Reading	10	100	25	Temperature	365	0
45_1	CMS Inflow Forecast	Inflow forecast normal	0	0	0	None	365	1
29_4	Dam Access	Typical access time	2	4	2.5	None	365	1
42_2	Penstock	Normal operation	0	0	0	None	365	0
48_3	Site Staff Availability	Staff available	0	0	0	None	365	0

Table 4-4: Database information from example scenario: Component Level

Identifier	Reservoir Level Name	Component Level Name	Operating State Name	Min	Max	Avg	Causal Factor Name	Min Date	Max Date
838_1	Turbine 1	Generator	Load Rejection	0.1	7	0.25	Maintenance	1	365
836_2	Turbine 1	Head Cover	Normal	0	0	0	None	1	365
1359_1	Gate 1	Gate opening	Normal	0	0	0	None	1	365
1360_2	Gate 1	Components failing in place	Components of the gate fail causing it to remain in place	0.5	120	7	Maintenance	1	365
1361_1	Gate 1	Components failing open	Normal	0	0	0	Normal	1	365
1362_1	Gate 1	Components failing closed	Normal	0	0	0	Normal	1	365

In the example scenario, the PLC/RTU is offline due to a voltage fluctuation, and the grid is offline because of a wind storm. Temperature fluctuations have affected the reservoir elevation sensor which is giving a false reading. There is also a load rejection which results in the unit being offline. Components of the gate are also failed due to a lack of maintenance and the gate is stuck in its current position. These tables provide a good indication of the information that the simulation model reads in to run the simulation: The minimum, maximum and average impact magnitudes and the causal factor date restrictions are used in the Monte Carlo generation of parameters for each iteration of the particular scenario being run.

A description of the system dynamics model developed for the simplified system is provided in the following section.

4.4 Simplified System Model Description

The simplified system as described in the database was initially modelled using the system dynamics software Vensim (Ventana Systems 2015) interfaced with Python (Python Software Foundation, 2012). It was eventually converted into a pure-code Python script using the *sdpy* package to facilitate simulation using cluster computing on the Linux operating systems at Compute Canada. Converting the code to a Python environment also significantly improved the simulation efficiency by reducing the overhead associated with passing information between Vensim and Python. Appendix E contains the Python script for the scenario generation. A complete package that can be used to run simulations can be found in the electronic files under the dam_safety_simulation folder. This section describes the model in more detail and provides the model testing results that compare the simulation outputs to the historically observed Cheakamus System (on which the model is loosely based). A description of the base case and the dam safety improved case for the simulations is also provided.

The key benefit of using a system dynamics software package such as Vensim is that the system structure can be constructed in an object-oriented way, allowing for easy visualization and modification of the relationships between system components. Subscripting is another useful feature. Vensim allows for multiple sectors or model views, which are related to one another using “shadow variables” that link the variables between the sectors. One drawback associated with Vensim and similar software packages is there may be limitations to the complexity of functions defined within the software. As a result of this limitation, a link between the Vensim program and Python programming language was made using the *venpy* package (Breach 2015) which allows function equations to be programmed directly in python. While this link is useful for model development and testing, there is a significant amount of overhead associated with passing information between the two programs. Since the goal of this research is to simulate the full suite of

potential scenarios, the model was eventually converted directly to Python script, however the model structure remains the same. The object-oriented building blocks and equations in Vensim and Python can easily be translated to the pure Python environment using the *pysd* package (provided in the dam_safety_simulation folder of the electronic appendix).

The following sections describe the system dynamics model development. Screen shots of the system structure are taken from Vensim.

4.4.1 Model description

The following sections provide the detailed equations used in each of the system dynamics model sectors. The model sectors follow the generic control loop of Leveson (2011), which is expanded on to detail the relationships modelled in each sector in Figure 4-7. The Hydraulic System State sector contains the water balance and pertinent relationships to that. Reservoir inflows, storage and outflows are modelled. Outflows are a sum of flows through the turbine, uncontrolled flows through the penstock, gate flows, overflows and dam breach flows. Dam breach initiation and gate blockage are also modelled within the sector, as well as the binary position of the power intake gate. The Sensors Sector includes the collection and relay of reservoir level information for use in the operations sector. Reservoir sensor errors and relay issues are also modelled within the sector. The Operations Sector models the decision making and implementation. This includes inflow forecasting, operations planning, remote or manual actuation and delays in mobilization of personnel to the site. The Gate Actuators Sector models the gate position and availability, which is a function of the condition of the gate components as well as power supply. The Turbine Actuators Sector models the condition of power flow release components and determines the releases through the unit and uncontrolled releases through a ruptured penstock or failed head cover. Finally, the Disturbances Sector models the implementation of adverse operating states (which are a model input). This includes failures, errors and delays as well as capacity losses at the gate due to debris accumulation.

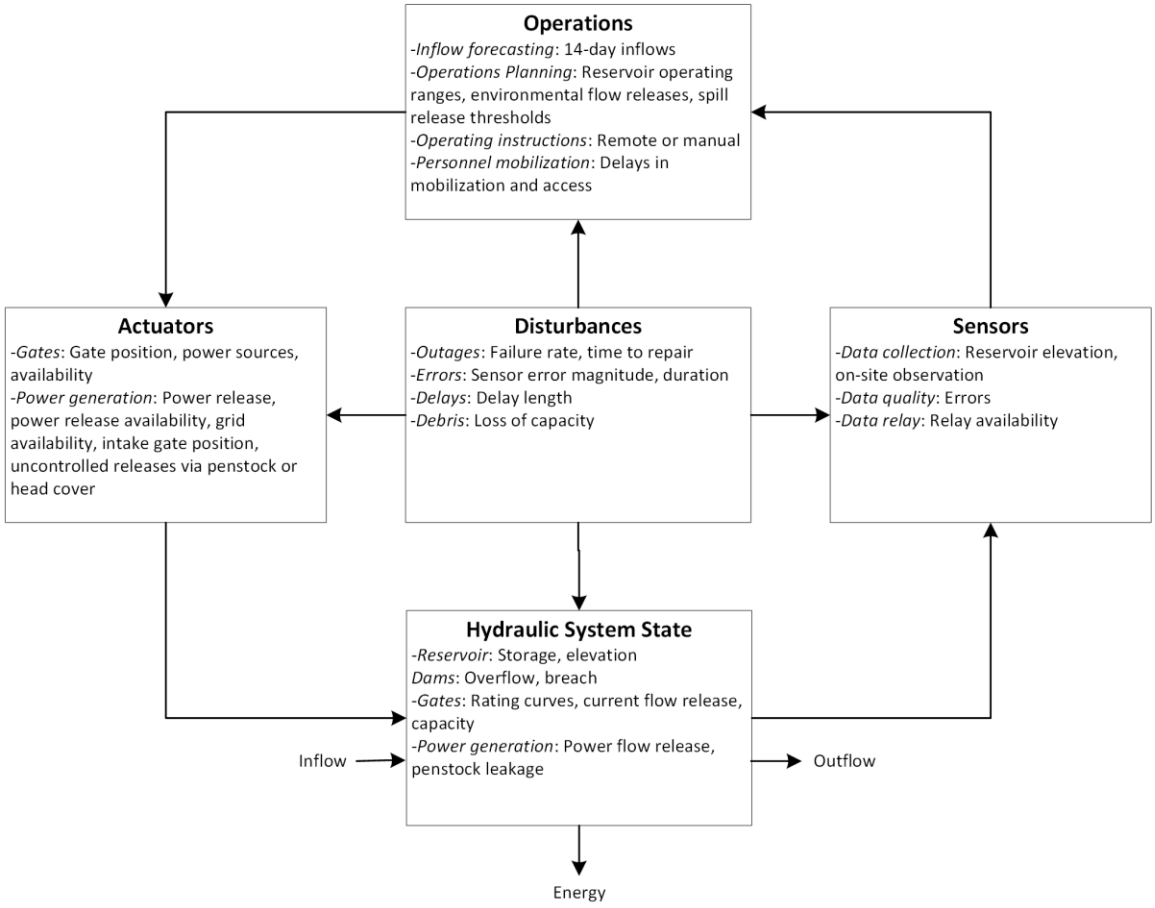


Figure 4-7: Simulation model sectors

The following paragraphs provide the stock-and-flow diagrams, details and equations for each of the model sectors. Each stock-and-flow diagram shows the relationships within the sub-system. Variables can also enter the sub-system from other sub-systems.

4.4.1.1 Hydraulic System State Sector

The Hydraulic System State Sector is shown in Figure 4-8 with the variable names and symbols shown in Table 4-5. This sector represents the status of the hydraulic infrastructure in the system relating to water retention (dams) and conveyance (water passages). It should be noted that components which move – such as gates, valves, and turbines – are considered Actuators. The functioning of these electrical, mechanical and structural components are represented within the Actuators sector and are not modelled as part of the Hydraulic

System State sector. Reservoir storage, flow conveyance through gates and turbines, overtopping and breach are represented in this sector.

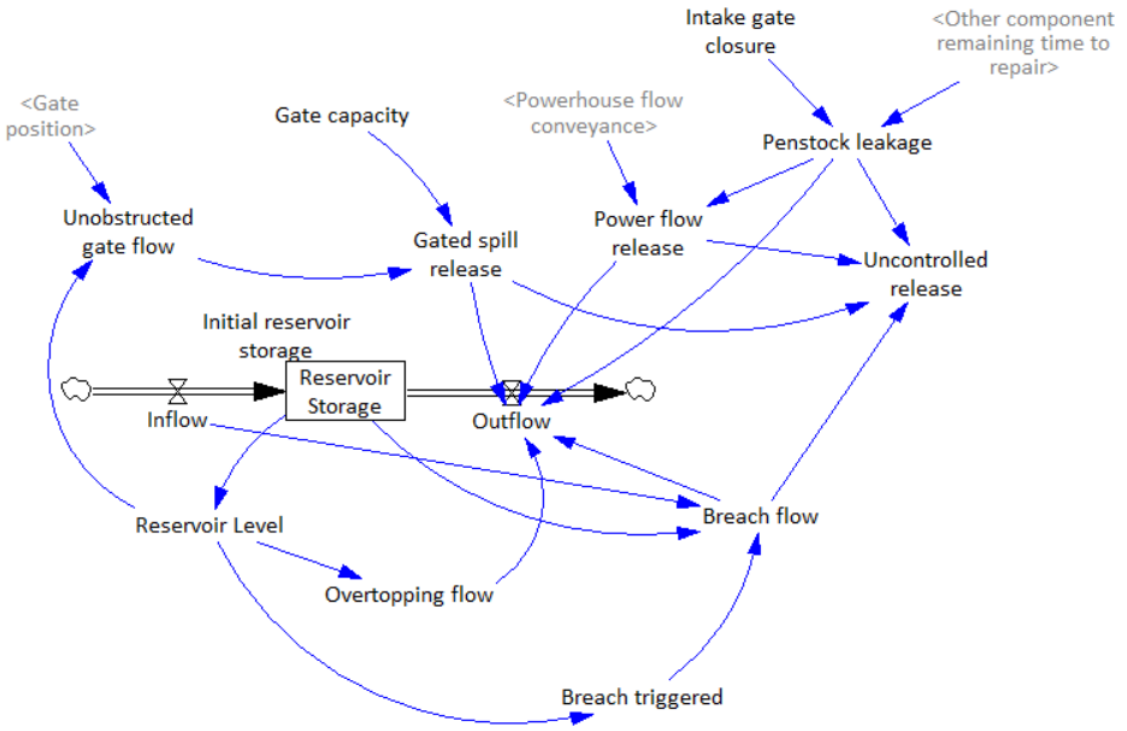


Figure 4-8: Hydraulic System State Sector

Table 4-5: Hydraulic System State Sector variable names

Variable name	Variable symbol
Reservoir Storage (m^3/s-day)	*S*
Reservoir Level (m)	*El*
Inflow (m^3/s)	*I*
Outflow (m^3/s)	*O*
Unobstructed gate flow (m^3/s)	*QGU*
Gated spill release (m^3/s)	*QG*
Power flow release (m^3/s)	*QP*
Gate position (m)	*g*
Gate capacity (%)	*GC*
Overflow (m^3/s)	*QOF*
Breach triggered (binary)	*EDB*
Breach flow (m^3/s)	*QDB*
Powerhouse flow conveyance (m^3/s)	*PQC*
Penstock leakage (m^3/s)	*QPen*
Other component time to repair (penstock) (days)	*Pttr*
Intake gate closure (binary)	*IG*

In this sector, reservoir storage is represented as a stock, with flows Inflow, I, increasing its value and Outflow, O, decreasing its value. The reservoir storage stock value is calculated by determining the difference in inflow and outflow at each time step, as shown in Equation 4:

$$\frac{dS}{dt} = I - O \tag{4.1}$$

Where *S* represents storage, *t* represents time, *I* represents inflow and *O* represents outflow. Storage is directly related to reservoir elevation (*El*) as described by the stage-storage table for the reservoir of interest. The Stage-storage curve (SSC) which determines reservoir elevation *El* from storage *S* and it's reverse (SSCRev) are supporting functions described in more detail in Appendix E.

The model outflow *O* represents a summation of the outflows from each of the *N* spillway gate conduits (QG_i, $i=1....n$), flows passing through the turbines (*QP*), and any uncontrolled flow releases (*UCR*). Uncontrolled flow releases include additional outflows from penstock leakage (*PL*), overflows (*OF*), and dam breach flows (*DBF*). Equations 5 and 6 pertain to model outflow (*O*) and Uncontrolled flow releases, respectively:

$$O = QG + QP + UCR \tag{4.2}$$

$$UCR = QPL + QOF + QDB \tag{4.3}$$

Unobstructed gated spill releases (*UGO*) are a function of reservoir elevation (*El*) and spillway gate position (*g*), as determined by the spill release rating curve. This function retrieves the value of the reservoir level and the gate position and calls the supporting function "GateFlowCalc" using those as arguments (See Appendix E).

In some operating scenarios, debris may block the spillway gate opening, reducing the capacity of the spillway gate, so the unobstructed gated spill release is then multiplied by the gate's real time capacity, (*GC*), to get the actual gated spill release (*QG*). This is shown in Equation 7:

$$QG_i = GC * QGU \tag{4.4}$$

Where *GC* is a ratio of full capacity and has a value between 0 and 1.

Overflow (*QOF*) is determined following Figure 4-9 using the overflow stage-discharge curve *OTC* as well as the stage-storage curve *SSC* and its reverse *SSCrev* from the supporting functions in Appendix D. The overflow stage-discharge curve represents the hydraulic relationship between the elevation of the reservoir and the total overflow discharge (through the free overflow spillways, as well as any additional discharges over the concrete and earthfill dams). The stage-discharge curve *OTC* is manipulated in the base case by increasing the spillway crest by 2m and multiplying the result by 0.3 to represent a scaled down capacity of the free overflow structures in the base case.

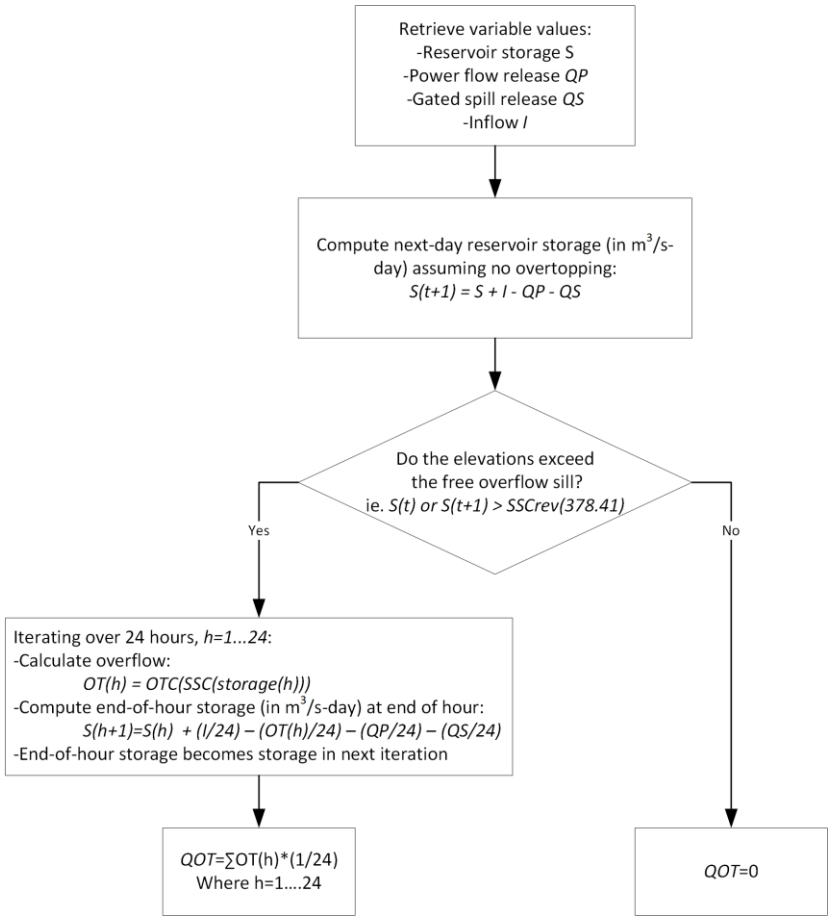


Figure 4-9: Overflow calculation

Because the Cheakamus Reservoir is somewhat flashy (the reservoir can fluctuate relatively quickly), the daily time step introduces some issues in calculating the aggregated

daily overflow spill. Because the fluctuations in reservoir level can occur at a finer time step than daily, the overflow values at the start of the day may not be equal to the overflow values at the end of the day. To address this problem, a nested hourly calculation is used, as shown in Figure 4-9. This takes into account whether the reservoir will exceed or drop below the free overflow spillway within the 24-hour period, and the resultant changes in overflow spill based on the fluctuating reservoir elevation.

Dam breach is assumed to be triggered (*EDB*) once the reservoir elevation exceeds a particular level (*DBEl*) above the earth dam crest (defined using expert judgement), and takes on a value of 0 for not breached or 1 for breached (Equations 8 and 9), with dam breach flows (*DBF*) equal to the full reservoir storage (the reservoir is completely emptied when the dam breaches):

$$For\ (El > DBEl): EDB = 1, \quad For\ (El < DBEl): EDB = 0 \tag{4.5}$$

$$For\ EDB = 1: QDB = S, \quad For\ EDB = 0: QDB = 0 \tag{4.6}$$

Penstock rupture is initiated through the Disturbances Sector when the penstock fails, which is represented by Other components time to repair with subscript Penstock, $Pttr$. Penstock leakage, Q_{pen} is equal to the "head cover max flow" from the turbine actuators sector (see Section 4.4.1.5), unless the intake gate is closed. This is described in Equation 4.7:

$$\begin{aligned} if\ IG = 0\ and\ Pttr > 0: \quad & Q_{pen} = HCMF \\ else: \quad & Q_{pen} = 0 \end{aligned} \tag{4.7}$$

The variable intake gate (*IG*) represents the status of the maintenance gate at the top of the penstock, where zero represents an open gate and 1 represents a closed gate. Power intake gates are present in most dam systems with hydropower generation at the upstream end of the power flow conduit. The intake gate provides a means to dewater and inspect/maintain the penstock and powerhouse components. In some dam systems, these gates may be able to close under excessive flows resulting from penstock rupture or head cover failure, reducing the negative impacts. In other systems, the reservoir must be at an elevation below

the sill of the intake gate before it can close. In the case of penstock failure, all of the water moving towards the powerhouse exits the penstock before making it there. Power flow releases from all units (*QP*) are equal to the powerhouse flow conveyance *PQC* less the water escaping through the penstock (*QPen)* via leakage or rupture as shown in Equation 4.8.

$$\begin{matrix} QP = PQC - QPen & if\ IG = 0, \\ QP = 0 & if\ IG = 1 \end{matrix} \tag{4.8}$$

If the intake gate is closed (*IG*=1), power outflow (*PO*) is equal to zero.

Variables shown in Figure 4-8 in grey font with chevron brackets are known as "shadow variables" These are the key variables which connect into the Hydraulic System State Sector from other sectors of the model. The variable "Other component remaining time to repair" enters the Hydraulic System State Sector from the Disturbances Sector. Variables "Gate position" and "Powerhouse flow conveyance" enter into the Hydraulic System State Sector from the Actuators Sector, which is broken down into Gate Actuators and Turbine Actuators.

4.4.1.2 Sensors Sector

The sensors sector is shown in Figure 4-10 and the variable symbols are outlined in Table 4-6.

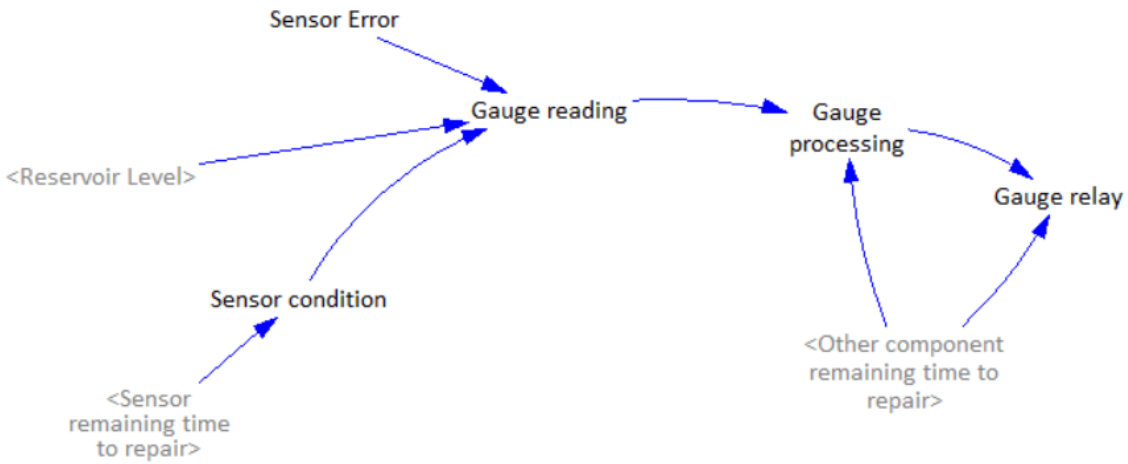


Figure 4-10: Sensors Sector

Table 4-6: Sensors Sector variable names

Variable name	Variable symbol
Reservoir level (m)	*El*
Sensor Error (%)	*SE*
Sensor condition (binary)	*SC*
Gauge reading (-)	*SRd*
Gauge processing (-)	*SP*
Gauge relay (-)	*SRl*

If the gauge is functioning properly (*SC*=1) then the gauge reading is equal to the reservoir level (*El*). If the gauge is failed (flat-lined), the value is equal to the last read reservoir elevation as per Equation 4.9:

$$\begin{aligned} For\ SC = 1:\ \ SRd &= El + El * \left(\frac{SE}{100}\right) \\ For\ SC = 0:\ \ SRd &= -9999 \end{aligned} \tag{4.9}$$

The gauge processing (*SP*) represents the interpretation step in the data collection system, which is carried out by software (a PLC) and will be site-specific. If the PLC is non-functional the *SP* will return a value of -9999 which indicates a missing value. The processed value is transmitted to the controller through the gauge relay (*SRl*) if the relay is available. The relay is usually carried out by a remote terminal unit (RTU), which is modelled as a single variable with the PLC. Thus, if the PLCRTU component is non-functional (*RA*=0), this also means that no information is transmitted to the controller Operations Sector.

4.4.1.3 Operations Sector

The Operations Sector for the hydropower system is shown in Figure 4-11 and Table 4-7 contains the relevant variables. This sector describes the use of information relating to the current state of the system to forecast inflows and make reservoir operating decisions.

Inflow forecasting may be done by applying a random, normally distributed error to the actual reservoir inflows, which are an input to the hydraulic system state sector. In reality, hydrologists use climate forecasts and watershed modelling to develop inflow forecasts that are considered during operations planning. While these processes could be incorporated into the simulation model, it would necessitate significant additional

computational effort. Instead, random errors may be applied to reservoir inflows to ensure operations planning is simulated using realistically inaccurate inflow information. Since the objective of this case study is to compare directly the base case and the dam safety improved case, inflow forecasting errors were removed from the potential operating states, since the random errors would differ between these two runs. Inflow forecast is simply the upcoming 14 days of inflows, as follows:

$$IF_d = I_{t+d} \tag{4.10}$$

where $d = 0, \ldots, 13$ and t is the current timestep.

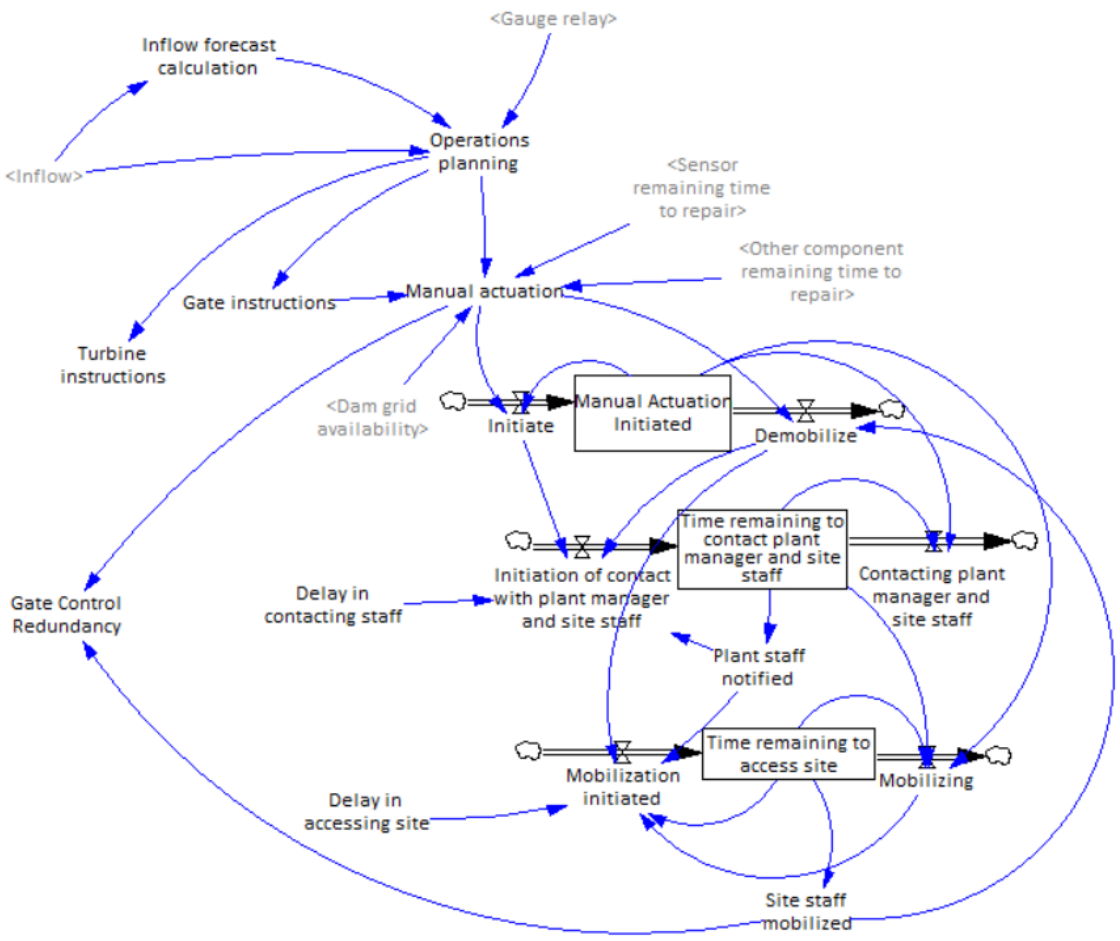


Figure 4-11: Operations Sector

Table 4-7: Operations Sector variable names

Variable description	Variable symbol
Inflow (m^3/s)	I
Inflow forecast calculation (m^3/s)	IF_d, $d=1…D$
Operations planning (m^3/s)	OP_f, $f=1…F$
Turbine instructions (m^3/s)	Ip
Gate instructions (m)	Ig
Gate control redundancy (-)	GCR
Other Component time to repair (-)	$OCttr$
Sensor time to repair (-)	$Sttr$
Manual actuation required (binary)	MA
Delay in contacting site staff	Ds
Delay in accessing site	Da
Contact initiated with site staff	CI
Contacting site staff	CS
Time remaining to contact site staff	TRC
Plant staff notified	PSN
Mobilizing initiated	$MobI$
Mobilizing	Mob
Site staff mobilized	SSM
Demobilize	$Demob$

Following inflow forecasting, operations planning (OP_f) proceeds. The result for operations planning is a vector of two variables (f=1…F and $F=K+N$), each representing a single instruction for a single controlled flow release component (K turbines and N gates, in this case study, $K=N=1$ and $F=2$). The main operations planning algorithm takes several key inputs (inflow forecast, reservoir elevation, day references, component availabilities and reservoir elevation limits) and determines the corresponding operating instructions for the system to ensure minimum flow releases are met and reservoir level restrictions are adhered to if possible. It can be found in the function OpsPlan which is described in Appendix E. The algorithm begins by assuming the minimum fish flow is released and the remainder of the inflow is passed through the powerhouse (up to the maximum) for a 14-day window from the current date. The resultant reservoir levels are then checked, adjusted and re-checked to ensure the operating instructions result in reservoir levels that are within the specified normal maximum (NMax) and minimum (NMin). To ensure enough water is available for the winter low-flow period, the normal minimum reservoir level was adjusted to El. 370 m for the months of November and December for the purposes of the modelling. Operations planning follows the algorithm shown in Figure 4-12, which includes power

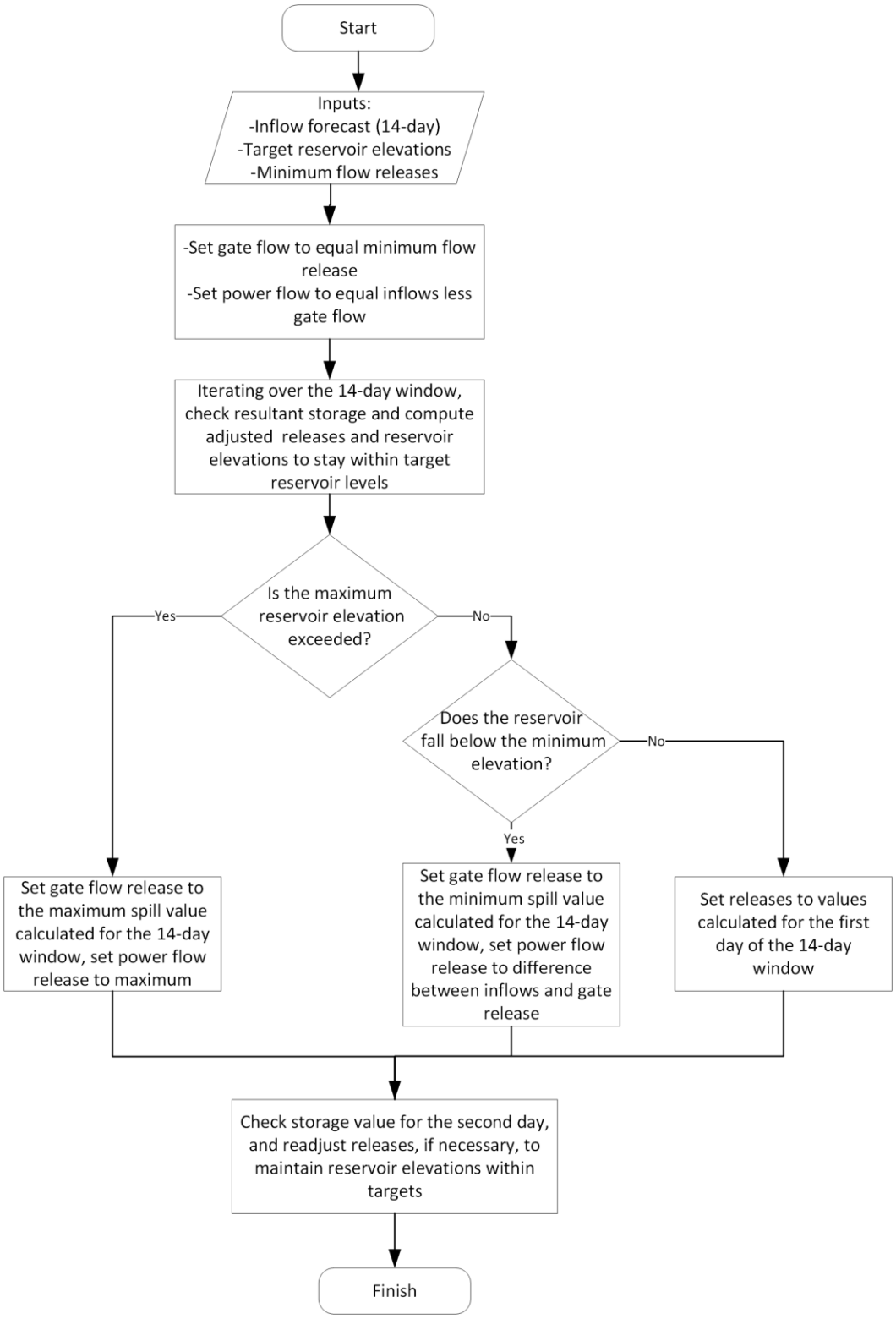


Figure 4-12: Operations Planning algorithm

flow releases through the turbine, following the logic that (a) fish flows are met, (b) additional inflows can be released through the power conduit, (c) any exceedance over NMax can be avoided by releasing more water through the power conduit or spill, and (d) any exceedance below NMin can be reduced from spill flows, then power flows. The algorithm generates instructions in terms of flow for the gate and turbine. The operations planning function in the Operations Sector OP_f collects and organizes the information necessary to be passed to the OpsPlan function which is described further Appendix E.

Gate operation may be carried out remotely or on-site. The default operation is remote, however manual actuation (*MA*) may be required if (a) communications equipment (PLCRTU) is out of service (*OCttr*>0) or (b) the reservoir elevation sensor is not functional (*Sttr*>0). The value of *MA* is set to 0 as the default, but changes to 1 if the equipment required to operate the gate remotely is failed, as per Equation 4.11:

$$\begin{aligned} MA = 0 \quad & if\ Sttr = 0\ and\ OCttr = 0 \\ MA = 1 \quad & if\ Sttr > 0\ or\ OCttr > 0 \end{aligned} \tag{4.11}$$

When *MA*=1, manual actuation is initiated. This occurs through a series of auxiliary variables and stocks which appear complex but are simple value holders that implement delays in contacting staff and mobilizing them to site.

The stock "Manual actuation initiated" (*MAI*) is a variable that, when equal to 1, indicates that the mobilization process is underway. The inflow to this stock is the variable Initiate, which is calculated as per Equation 4.12:

$$\begin{aligned} if\ MA = 1\ and\ MAI = 0: Initiate = 1 \\ Else: Initiate = 0 \end{aligned} \tag{4.12}$$

Demobilization (*DM*) is the outflow of the stock *MAI*, and sets this value back to zero when staff have mobilized and are on site (*SSM*) and manual actuation (*MA*) is no longer required, as per Equation 4.13:

$$\begin{aligned} if\ SSM = 1\ and\ MA = 0: DM = 1 \\ else: DM = 0 \end{aligned} \tag{4.13}$$

The next step in the process is notifying the plant manager so that site staff can be mobilized. This is represented using a stock "Time remaining to contact staff", (*TRC*), which tracks any delays in this process. The stock input is "Contact initiation" (*CI*) and the delay associated with the contacting and dispatch of staff, *Ds*, is an input from the disturbances sector. Plant staff notified (PSN) is another variable that tracks whether staff have been made aware of any issues at the site. Contact initiation (*CI*) is calculated as follows:

$$if\ Initiate = 1\ and\ SSM = 0\ and\ PSN = 0: \quad CI = Ds$$

$$else: \quad CI = 0 \tag{4.14}$$

This sends a pulse to the *TRC* stock, which is equal to the predetermined delay time (if any), which is pre-determined at the start of the simulation through the Monte Carlo scenario generation. The stock outflow, "Contacting" (*CS*) is then equal to the timestep while the value of the stock is greater than zero. Once the *TRC* stock has filled and drained, the plant staff are considered to be notified *PSN*. The *PSN* variable represents this by taking on a value of 1 when the staff are dispatched to site, and zero when they are not, as follows:

$$\begin{gathered} if\ MA = 1\ and\ TRC = 0: PSN = 1 \\ else: PSN = 0 \end{gathered} \tag{4.15}$$

Next, the site staff begin to mobilize to the site. There may be a delay in mobilization due to site access issues such as traffic or emergencies, "delay in accessing site" (*Ds*). These delays are a direct Monte Carlo generated input from the simulation model when site access is delayed. This delay is again represented using a stock "Time remaining to access site", (*TRA*), which receives a pulse of inflow from "Mobilization initiated" (*MobI*), and has outflow "Mobilizing" (*Mob*). The variable *MobI* is calculated as follows:

$$\begin{gathered} if\ PSN(t) = 1\ and\ PSN(t-1) = 0: MobI = Ds \\ else\ if\ Demob = 1: MobI = 1 \\ else: MobI = 0 \end{gathered} \tag{4.16}$$

This variable sends a pulse equal to *Ds* (the delay time) when mobilization is initiated, and zero otherwise except during demobilization, when the standard mobilization time (1 day) is sent as a pulse to the stock *TRA* to re-set the standard site access time. The variable "Mobilizing" (*Mob*), drains the stock *TRA* at the rate of time, when its value is positive – this represents the travel of site staff to the dam. Finally, once the value of the *TRA* stock is zero and mobilization is still required (*MAI*=1), the site staff are mobilized and at the dam "Site staff mobilized" (*SSM*=1), as per Equation 4.17:

$$if\ TRA = 0\ and\ MAI = 1: \quad SSM = 1$$

$$else: \quad SSM = 0 \tag{4.17}$$

Once the site staff are mobilized, actuation of the gate can occur manually. Demobilization (*Demob*) occurs when manual actuation is no longer required (*MA*=0) and site staff are present at the site (*SSM*=1), as follows:

$$if\ MA = 0\ and\ SSM = 1: \quad Demob = 1$$

$$else: \quad Demob = 0 \tag{4.18}$$

4.4.1.4 Gate Actuators Sector

The Actuators Sector has been broken down into two sub-sectors: (1) Gate Actuators and (2) Turbine Actuators, because both the function and purpose of these components are very different. Outlet Gates may be operated manually or remotely and rely on either grid power or a backup power source as well as a series of interconnected mechanical and electrical components which function together to make the gate operable. Turbines are typically operated remotely, require an operational grid to be functional (power must be exported somewhere) and rely on vastly different components to achieve their intended purpose. As such, actuation of a gate is not modelled alongside actuation of a turbine and the sectors are shown separately.

The Spillway Gate Actuators Sector is shown in Figure 4-13 with relevant variable symbols presented in Table 4-8. This sector represents each of the mechanical, electrical and structural components involved in operation of a spill release gate. The components are

grouped based on the outcomes of component failure into three categories: (1) Components failing the gate in the closed position, (2) Components collapsing the gate and (3) Components causing the gate to fail in its current position. The model has been generalized as much as possible to represent both radial and sluice-type spillway gates but may need to be modified for representation of different types of gates or for valve release facilities (eg. Stop-log gates, Hollow cone valves, Howell-Bunger valves). Backup power supplies may also easily be added to the model.

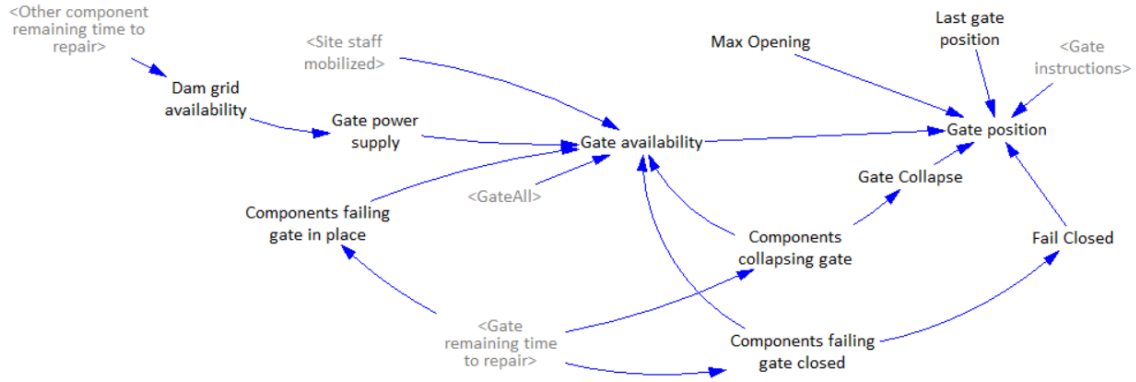


Figure 4-13: Gate Actuators Sector

Table 4-8: Gate Actuators variable names

Variable Description	Variable Name
Gate instructions (m)	*Ig*
Site staff mobilized (binary)	*SSM*
Gate position (m)	*g*
Last gate position (m)	*Lg*
Gate availability (binary)	*GAv*
Gate remaining time to repair (days)	$GRTTR_c$, $c=1...C$
Gate power supply (binary)	*GPS*
Gate collapse (binary)	*GC*
Gate failed closed	*FC*
Gate failed in place	*FIP*
Maximum gate position	*MGP*

Failures of the component groups are each associated with different times to repair which are modelled in the “Disturbances” sector. Disturbances in the system, for example seismic events, may affect all or some of these components and the maximum repair time for each

of the affected components then becomes the length of time the gate is unavailable (out of service) for.

The gate can either be operated remotely or by site staff if manual actuation (*MA*) is triggered as described in the previous section. If *MA*=1, site staff must be mobilized (*SSM*=1) in order for the gate control system to be operated and for actuation to take place. If *MA*=0, the gate's remote actuator is functioning properly and the gate may be operated from the control center.

The gate components are binary indicators of component availability and are used in the calculation of overall gate availability and as indicators of whether the gate is collapsed (*GC*) or failed in place (*FIP*) or failed closed (*FC*). The values of the *C* affected gate components for each gate, *i*, are set to 0 if the remaining time to repair is greater than 0 and 1 if the remaining time to repair is 0 (ie. there is no damage to the component), as per Equation 4.19:

$$For\ GRTTR_c > 0: c = 0\ ,\quad For\ GRTTR_c = 0: c = 1 \tag{4.19}$$

Gate availability is set equal to one if all gate components are available (values equal to one), the power supply is available (*GPS*=1) and either remote actuation is possible (*MA*=0) or staff are on site to operate the gate (*MA*=1 and *SSA*=1). Gate collapse (GC_i) is set equal to one if the "components collapsing gate" is equal to zero and fail closed (FC_i) is set equal to one if the "components failing gate closed" is equal to zero. Gate instructions are measured in meters of opening and are determined from the Operations Sector, entering the Gate Actuators sub-system as a shadow variable. Gate position is then determined as follows.

$$\begin{aligned} g_i &= MGP \quad for\ GC = 1 \\ g_i &= 0 \quad for\ FC = 1 \\ g_i &= Ig \quad for\ GAv = 1 \\ g_i &= Lg \quad for\ GAv = 0 \end{aligned} \tag{4.20}$$

Last gate position *Lg* is stored by Python and used as the default gate position if the gate is unable to be moved due to failure of a component. The gate position *g* is then used as an input to the Hydraulic System State sector.

4.4.1.5 Turbine Actuators Sector

The model structure for the power Actuators Sector is shown in Figure 4-14 with relevant variables described in Table 4-9. The power Actuators Sector has been simplified significantly due to the high complexity associated with operation of a generating unit. Wicket gates could be modelled as a stock with flows of opening and closing, however this would require modelling the governor and other turbine components in significant detail. It was assumed that modelling in this way would increase complexity but not improve the result significantly. As such, powerhouse flow conveyance is the key variable being modelled, and the availability of the components required for the powerhouse to function are shown as inputs that determine whether power can pass through the powerhouse and electricity can be generated (Power remaining time to repair). The two very high-level power component failures that are being modelled are the generator (which causes a load rejection) and the turbine head cover which can fail causing an uncontrolled release of water into the powerhouse and downstream. In reality, there are many ways in which a turbine can fail to operate safely, however the inability to pass flow and the uncontrolled release of flow are the two major outcomes of significant power related failures, so these components were considered to be representative.

The values for head cover (*HC*) and generator (*PGen*) are determined by "Power remaining time to repair" which tracks the time left in repairs following failures of these components. If the remaining time to repair value is positive, their value is set to zero (this equation is the same as for the gate components above). Unit availability then depends on the availability of the wicket gates, generator and grid (*GrAv*) following Equation 4.21:

$$\begin{aligned} if\ HC = 1\ and\ PGen = 1: \quad PUA = 1 \\ else: PUA = 0 \end{aligned} \tag{4.21}$$

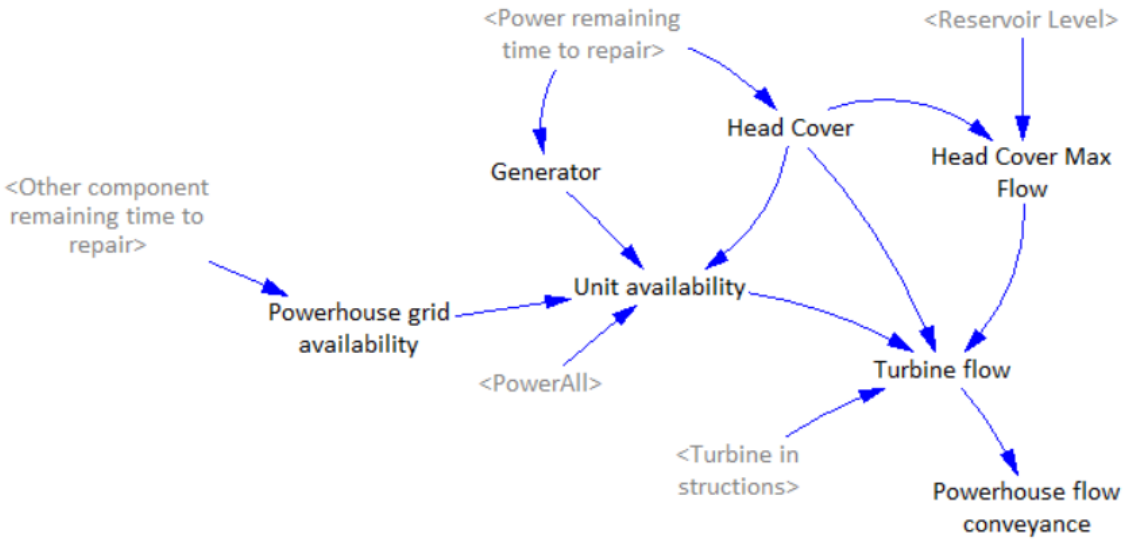


Figure 4-14: Power Actuators Sector

Table 4-9: Power Actuators variable names

Variable description	Variable name
Turbine instructions (m^3/s)	*IP*
Unit availability (binary)	*PUA*
Unit flow (m^3/s)	*QU*
Generator (binary)	*PGen*
Head cover (binary)	*HC*
Powerhouse grid availability (binary)	*GrAv*
Powerhouse flow conveyance (m^3/s)	*PFC*
Head cover max flow (m^3/s)	*HCMF*
Intake gate closed (binary)	*IG*

If any of *PWG*, *PGen* or *GA* are equal to zero, *PUA*=0 and the unit cannot release any water (*QP*=0) unless the head cover (*HC*) is failed, in which case the maximum head cover flow is released through the unit, as per Equation 4.22:

$$\begin{aligned} If\ PUA = 1: &\quad QP = IP \\ if\ PUA = 0\ and\ HC = 1: &\quad QP_j = 0 \\ IF\ PUA = 0\ and\ HC = 0, &\quad QP = HCMF \end{aligned} \tag{4.22}$$

Head cover max flow (*HCMF*) is a site-specific relationship to be determined by the modeller. In this case, the assumption is that the maximum turbine flow for the current reservoir level can be multiplied by five to represent the total flow that would pass through the failed unit. If this flow causes reservoir elevations to drop below the sill, a correction

is made to represent the reduction in flow being passed into the power tunnel from the reservoir (*Qsill*). The head cover release also depends on the intake gate closed variable (*IG*), and is equal to zero if the intake gate is closed. The equation is as follows:

$$if\ IG = 0: \quad HCMF = \min(5 * TQ_{MAX}(RSE, av = 1), Qsill)$$
$$else: \quad HCMF = 0 \tag{4.23}$$

Where TQ_{MAX} represents the supporting function described in Appendix E which calculates the maximum flow through the turbine for a given reservoir level (*RSE*) and availability set equal to 1. *Qsill* represents the reduction in this value that would be observed when the reservoir passes below the sill within the current day. A simple volume calculation is done to calculate *Qsill* as follows:

$$Qsill = \max\ (S + I - QG - SSCrev(363.06), 0) \tag{4.24}$$

where S is the storage, I is the inflow QG is the gated spill release, and SSCrev represents the reverse lookup from reservoir level to storage. El. 363.06 is the elevation of the gate sill. *Qsill* cannot be negative. It is important to note here a key difference between the base case and the dam safety improved case. For the dam safety improved case, the head cover maximum flow, $HCMF$, is multiplied by 1/24 to represent intake gate closure within an hour of rupture occurrence. This is because the time-step of the model is daily and it is assumed that the gate closure would happen immediately upon detection of the rupture (within one hour), so the maximum flows are simply scaled by this factor.

Equation 18 shows that the powerhouse flow conveyance (PQC) is equal to the sum of releases through each turbine:

$$PQC = QU \tag{4.25}$$

Powerhouse flow conveyance connects into the Hydraulic System State Sector.

4.4.1.6 Disturbances Sector

The main goal of the Disturbances Sector is to implement component failures which result from a variety of causes from the components operating state database. Components have

been divided into four groups: Gate components, power components, other components and sensors. This helps facilitate detailed modelling of various component failures, for example the gate hoist or the gate motor becoming out of service following some disturbance. Other components include the penstock, communications equipment, and the grid. The gate and power components include all key components of the "Gate actuators" and "Turbine actuators" sectors, respectively, which may fail resulting in various impacts to the system. Stocks are used in this sector to represent the remaining time left on the repair. The stock inflows consist of a single pulse (incoming time to repair). The stocks are drained by a constant time when their value is positive, as shown in the following equation:

$$\begin{aligned} &if\ cRTTR > 1:\ cRepair = 1 \\ &else\ if\ \ 0 < cRTTR < 1:\ cRepair = cRTTR \\ &else: \qquad cRepair = 0 \end{aligned} \tag{4.26}$$

This ensures the stock is drained by time when its value is positive and prevents the stock value from becoming negative. The small c represents the component type (Gate, Power, Other or Sensor). The component failure variables connecting to the time to repair stock inflows receive information from the model to implement component failures of various lengths at specific time steps (the Monte Carlo inputs). This is demonstrated in Section 3.4.1 and has been generalized to take Monte Carlo inputs of Impact Time IT and Impact Length IL:

$$\begin{aligned} &if\ t = IT:\ cComponent\ Failures = IL \\ &else:\ cComponent\ Failures = 0 \end{aligned} \tag{4.27}$$

GateAll and PowerAll represent the total maximum remaining time to repair of all components represented by the stock, as indicators that are used in the Gate Availability and Turbine Availability calculation. The component Remaining Time to Repair ($cRTTR$) values for Gate, Power, Sensor and Other components are then routed into the model to the corresponding location to be implemented in simulation, as described in the previous model description sections.

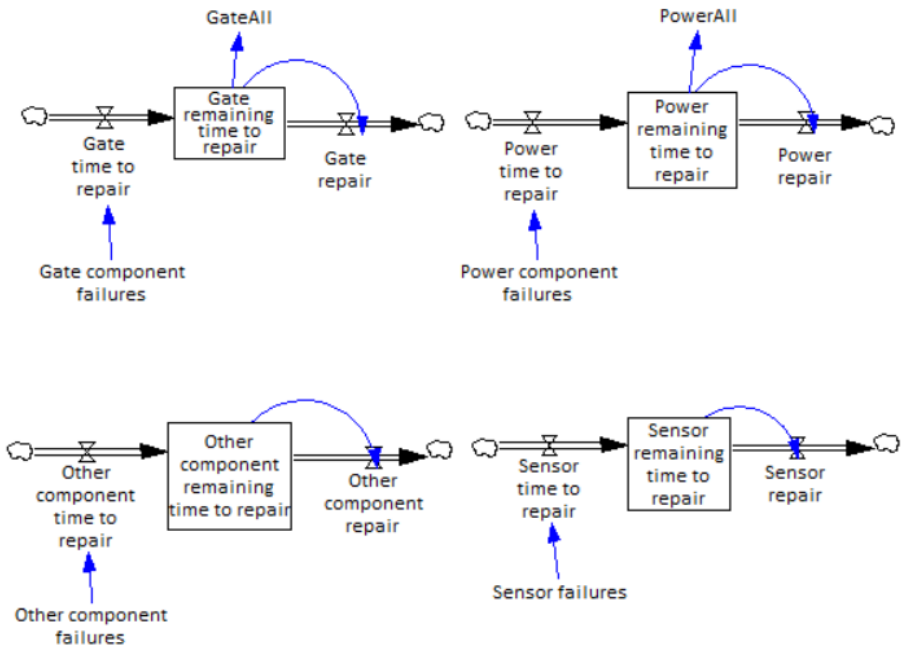


Figure 4-15: Disturbances Sector

4.4.2 Simulation model testing

As discussed in Section 3.4.3, testing can be done to gain confidence in the model performance. Checking the water balance to ensure the formulae are properly defined is an important step. Another very useful test is to compare the model outputs with the observed data for the system. This was done by running the simulation using Cheakamus historical inflows and comparing the results with the real Cheakamus operating data. The model was tested by modifying the operations planning function and comparing the results with the historical data for operations including reservoir elevation, turbine flow and spill. The results of the model test are shown in Figure 4-16.

Figure 4-16(a) and (b) contain the reservoir elevations from the observed record and simulation, respectively. It is clear from the plots that the simulation model tends to hold the reservoir higher than it would be under typical operation. This is a result of the operations planning algorithm, which does not use optimization. In the initial development of the complex model, operations planning was performed daily using a differential evolution optimization model. The optimization model planned reservoir levels for one

year of expected inflows using the 14-day inflow forecast and weekly average inflows for the remainder of the year. The change in reservoir storage was used to calculate the instructions for the gates and turbines. This procedure for operations planning results in a very accurate model test that is shown in King et al (2017). Once the scale of the simulation problem was more accurately defined, the optimization step was determined to be sufficiently time consuming to justify its removal from the program, and the model was switched to a simple algorithm to calculate releases. As such, the simulated reservoir levels for the historical model test are not as close to the observed values, however they are still well within the operational limits.

Figure 4-16 (c) and (d) show the turbine flows from the historical operations record in comparison to the simulation. The median lines are fairly close, though the simulation model tends to release more water than the historical record, which is likely due to the fact that the Cheakamus System is often used for peaking, meaning it may run fully during certain hours of the day and be shut off at night, resulting in lower overall flows.

Figure 4-16 (e) and (f) show total spill releases for the historical and simulated operations, respectively. There is a close agreement between the medians for spill release, however larger spill events tend to be reduced in the simulated results in favour of slightly longer, more moderate spills.

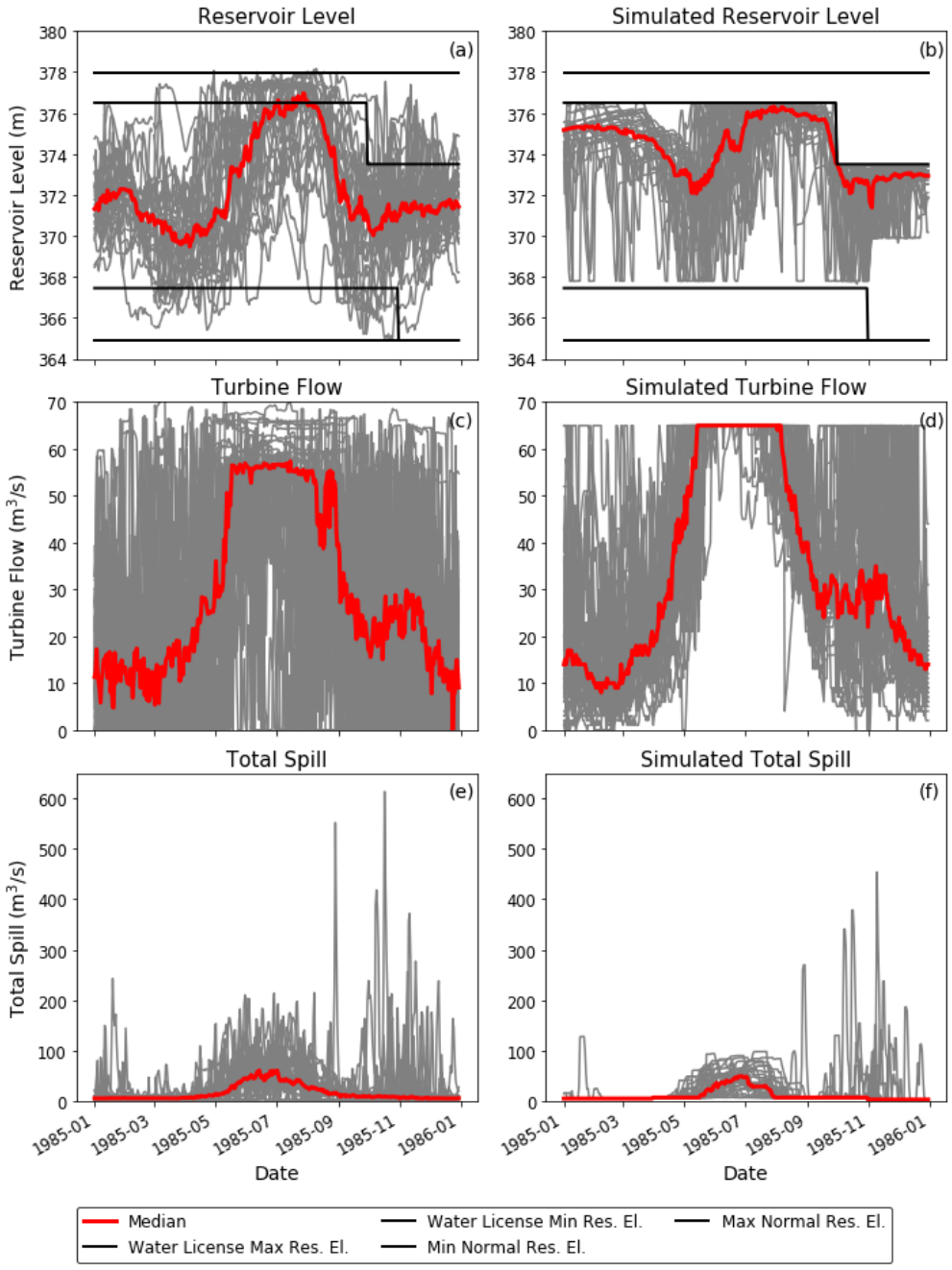


Figure 4-16: Operations validation for the simplified system model

4.4.3 Base case vs. dam safety improved model runs

Two full runs through the potential operating scenarios are performed in this research for two different cases: the base case and the dam safety improved case. The difference

between these scenarios is described in this section and summarized in Table 4-10. The goal of these two different runs is to show how the simulation model results can be used to assess improvements made by changing operating strategies and investing in upgrades to the system.

Scenario A: The base case is a simplified version of the Cheakamus system as described above. The key change that was made for this case was that the free overflow spillway capacity was significantly reduced (by 70%) and the crest was raised by 2m. The purpose of this was to directly induce more failures in the base case, creating a substantial difference between the dam safety improved case which has a free overflow spillway identical in size to the one in the real Cheakamus system. This change was made in response to a very low observed failure rate for the dam safety improved case, given the ability of the free overflow spillway to safely pass even large inflows when the capacity of the system to convey water through the gates and turbines is significantly reduced.

Scenario B: The dam safety improved case has a full-sized free overflow spillway consistent with that of the real Cheakamus Dam. In addition to this, the intake gate for the powerhouse is upgraded to allow it to close under penstock rupture or head cover failure flow. Because of the daily timestep and the relatively small and flashy Daisy Lake, this is implemented in two ways. First, the intake gate is closed immediately the day after a rupture or head cover failure is realized. Secondly, the total penstock rupture and head cover failure flows are reduced to $1/24^{th}$ of their actual values, to reflect closure of the gate within an hour of the initiating failure. Another key change in the dam safety improved model is that in the event of lowered capacity in the system resulting from a gate outage, or loss of remote control due to PLCRTU outage or grid outage, the target reservoir level is lowered to El. 367.8m which is 0.5m above the crest of the spillway. The goal of this operational change is to avoid free overflow events and dam failure by preparing for large inflow events which the system may not be capable of conveying through the power passage alone. Increased redundancy in the communications equipment was modelled by reducing the number of outages for the PLCRTU component to one half of the scenarios. This is done by modifying the Monte-Carlo generated outage times for a randomly selected half of the iterations to zero. Sensor errors and outages were similarly reduced by one half

to indicate improved sensory equipment at the site. Finally, the instances of the gate failing closed were reduced by 20% to reflect upgraded components in the gate resulting in fewer of these failures.

Table 4-10: Base Case vs. Dam Safety Improved Case

Base Case	Dam Safety Improved
Free overflow spillway restricted to release only 30% of Cheakamus Dam discharges, with a crest 2m higher	Free overflow spillway identical to Cheakamus Dam
Single PLC/RTU device	Dual PLC/RTU device, resulting in 50% fewer outages of that component
Intake gate unable to close under penstock rupture or head cover failure flows	Intake gate upgraded to allow closure under penstock rupture or head cover failure flows
Default gate redundancy	Gate redundancy improved to reduce instances of the gate failing closed by 20%
Reservoir level targets consistent with typical Cheakamus operations	Reservoir level target lowered to El. 367.8m if system capacity is restricted

Each scenario is run through the simulation model with two thousand iterations and the complete simulation is run once for the base case and once for the dam safety improved case. The goal of this is to illustrate the improvements made between the two runs. Because there are such a large number of scenarios and iterations being modelled, more varied inflow sequences are required than observed in the historical record. This is described in the following section.

4.5 Simulation Model Input Data

The simulation model data inputs include the physical relationships, the synthetic inflows and the baseline operations (reservoir levels) for the system.

4.5.1 Physical Relationships

The first physical relationship used in the model is the stage-storage curve, which relates the elevation of the reservoir to the storage in m^3/s-day. The units chosen to represent storage help simplify the calculations within the simulation model. The stage-storage data for the Cheakamus Project were used in the simulation model and are presented in

Appendix A. Curve-fitting was used to develop a relationship valid for the possible reservoir elevations, and the resultant relationship is described in Appendix E for the stage-storage curve (SSC).

The stage-discharge curve for the Cheakamus System are also used in the simulation model (See Appendix A). A function representing the total overflow is created using curve-fitting to reflect the overflow discharge pertaining to a certain elevation. The resultant function is described in Appendix E for the overflow curve OTC.

The combined gate rating curve for the two Cheakamus Spillway Operating Gates (SPOGs) is also used in the model. The rating curves for each gate are combined into a single curve for a larger gate by adding the discharge columns from the curve. The resulting combined curve is used directly in the model in a 2-dimensional interpolation.

The maximum turbine flow pertaining to different reservoir elevations is required in the model to ensure generating restrictions at low reservoir elevations are taken into account. This is calculated from the units operating curves and converted to a piecewise linear function. A similar piecewise linear curve is developed for the maximum possible gate flow at different reservoir elevations.

4.5.2 Synthetic Inflow Generation

Synthetic inflow generation was carried out by reshuffling and perturbing the historical climate data using a stochastic weather generator (KnnCAD) and using the results as inputs to the Raven hydrologic modelling tool. KnnCAD and Raven are described in Section 3.3.1. For the Cheakamus Hydropower Project, a single station located at the dam (CMS) is used for inflow forecasting.

Twenty-seven years of historical daily climate data from the CMS station was used as an input to the KnnCAD weather generator. The data included daily minimum and maximum temperatures as well as precipitation. KnnCAD reshuffles and perturbs the historical climate data to come up with a statistically similar block of data the same length as the

input data, so 371 blocks of climate data were created by the weather generator for a total of 10,017 years of data. For real applications of this approach, closer to 1 Million years of climate data is recommended to ensure adequate variability in the inflow sequences, however 10,000 was determined to be adequate for the purposes of this proof-of-concept example.

A validation of the historical versus simulated climate data is shown in Figure 4-17. Figure 4-17(a) and (b) contain boxplots of daily precipitation (no outliers) and total monthly precipitation respectively. The blue line plot overlaid on the boxplots shows the historical median values. For daily precipitation, there is a close match between the median historical and simulated values. The daily precipitation values were shown without outliers because the outliers were quite high in comparison with the boxplots, with one simulated value exceeding 800mm in March. The number of outliers in the data indicates the ability of the model to simulate more extreme precipitation events than in the observed record. The synthetic climate data tends to underestimate the total monthly precipitation, with the historical medians being close to the 75^{th} percentile of the simulated data in January, March through July, October and December. For February and August there is a close agreement and there is a smaller underestimation in September and November. Despite the underestimation of the median total monthly precipitation values, the simulated data does match the monthly trend shape and there are a fair number of outliers from the monthly data. Figure 4-16(c) and (d) contain monthly minimum and maximum temperature boxplots of the simulated data, with historical medians overlaid on the graphs. There is a fairly close agreement in the trends, however both the median minimum and maximum temperatures do tend to slightly underestimate the historical medians. There are, however, a significant number of outliers which indicates values outside of the historical record are present in the simulated data.

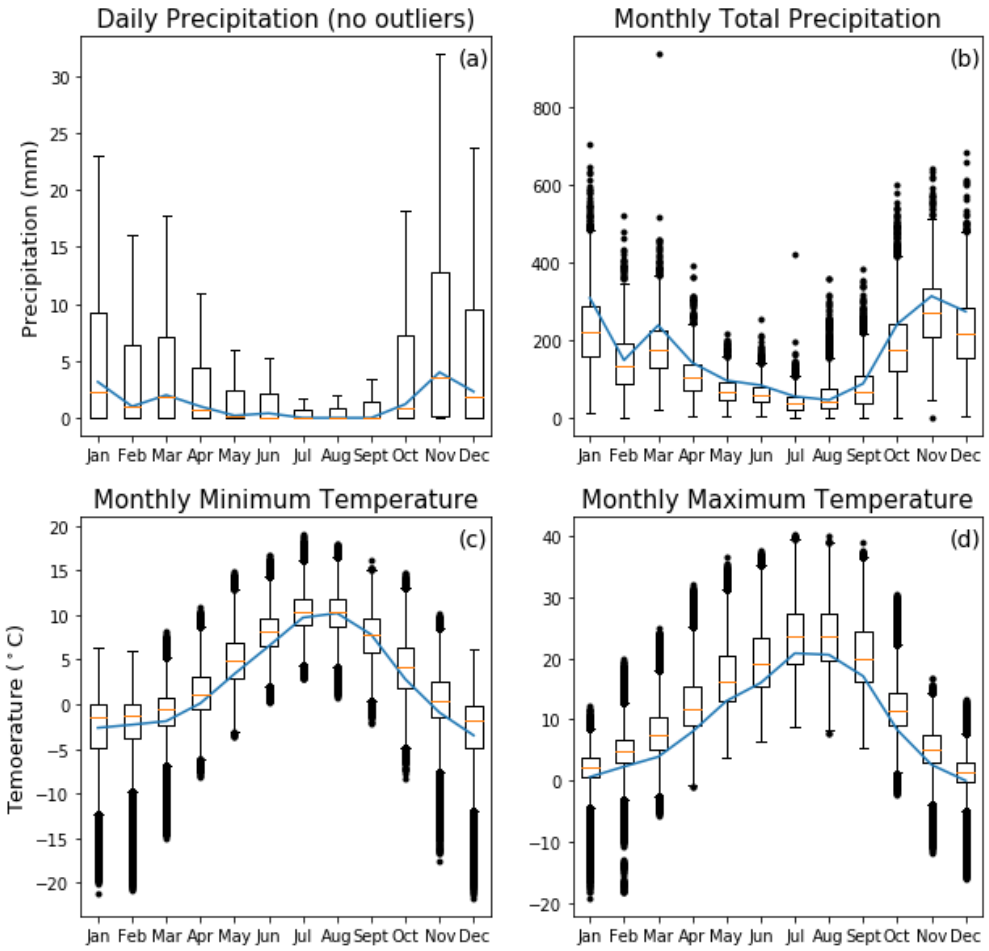


Figure 4-17: Validation plots for synthetic climate data at for CMS station

The UBC watershed model requires water-years as an input, which run from October 1 to September 31, so some reorganizing of the resultant datasets is required. Once the data is reorganized, it can be used as an input to the calibrated watershed model (UBC Watershed model on Raven) for the Cheakamus System. BC Hydro provided an up-to-date calibration for use in this research so the calibration and validation procedure for the hydrologic modelling is not discussed in this text.

Figure 4-18 (a) and (b) show the historical and simulated daily inflow data, respectively with the lightest blue lines showing the 10th and 90th percentiles, the medium blue lines showing the 25th and 75th percentiles and the 50th percentile shown in dark blue. The percentiles of the historical vs. simulated data align well and there are significantly more extreme inflow events observed in the simulated record, which is the goal of synthetic

inflow generation. The synthetic inflow percentiles are slightly smoother because of the large number of observations for each day (10,000). The maximum inflow event observed historically is approximately 650m^3/s and the maximum inflow event observed in the simulated record is about 2000m^3/s.

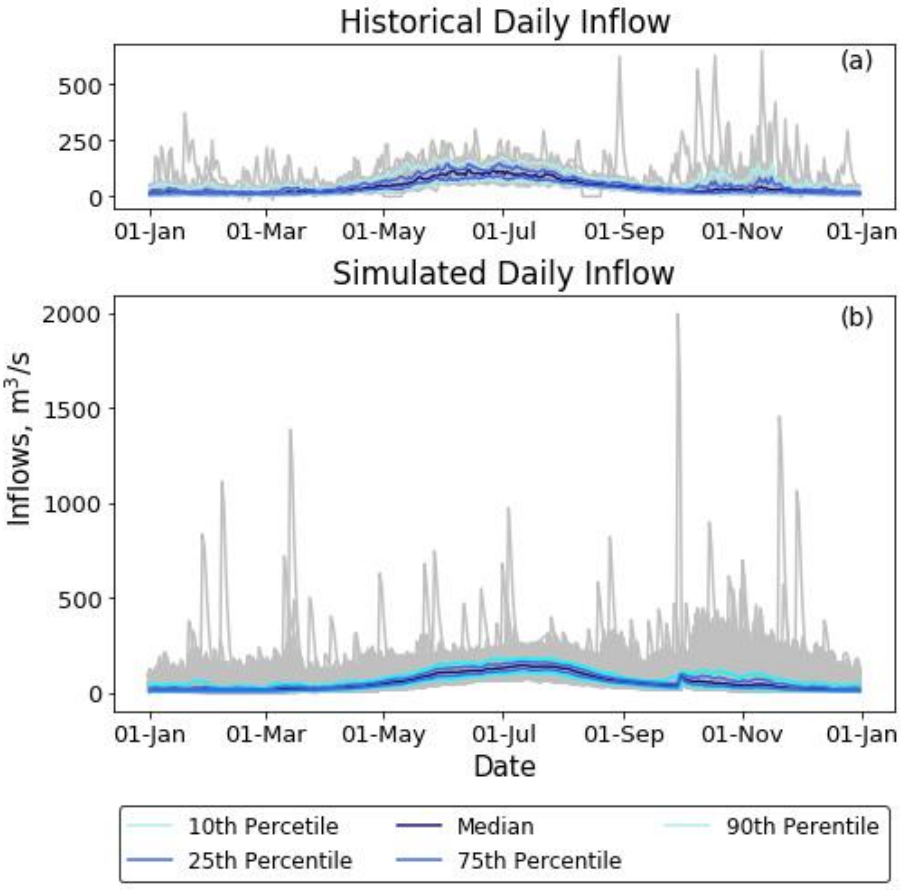


Figure 4-18: Validation of synthetic inflow sequences, Daisy Lake inflows

4.5.3 Baseline Operations Data

The baseline operations data were computed by running the simulation model for the 10,000 years of synthetic inflows and recording the observed reservoir elevations given that nothing within the system had failed. The results from the baseline operations data are shown in Figure 4-19. It is important to note that the operations planning algorithm in this simulation has perfect 14-day foresight about inflows and all flow release facilities are operational, so only one reservoir level excursion above the normal maximum level of El.

376.5 is observed over the 10,000 years. This excursion corresponds to a peak daily inflow of 2,000 m3/s. During this large inflow event, the spill capacity of the gates and power flow releases are exceeded and the reservoir increases to El. 380 m. This is below the elevation of the free-overflow spillway, which is at El 380.41 in the base case (the crest of the concrete dam).

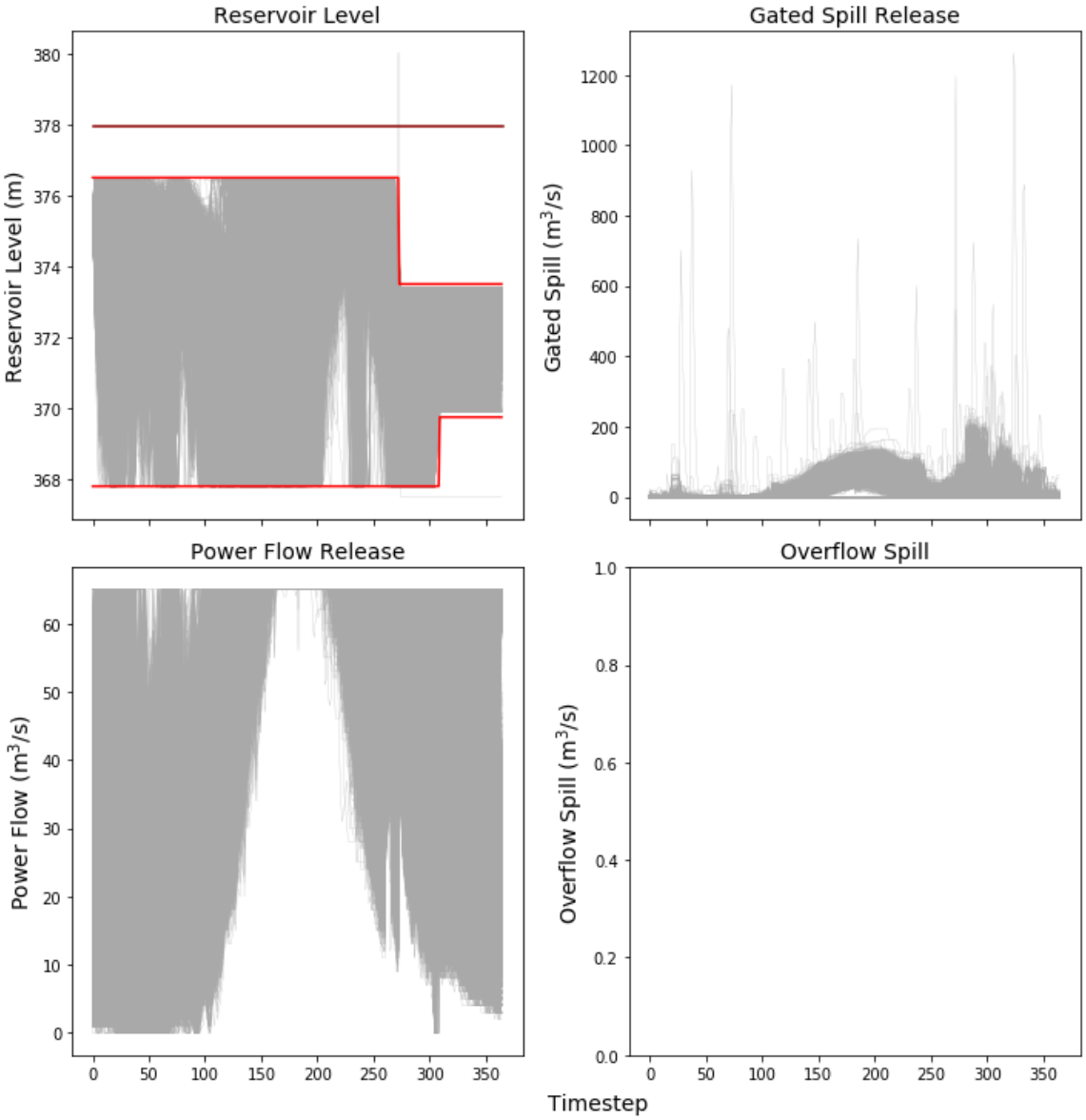


Figure 4-19: Baseline operations data from 10,000 year synthetic inflow record

4.6 Simulation Results

The results from the simulation are presented in the following sub-sections. There were a total of 552,960 scenarios run for the base case and the dam safety improved case. Each scenario was executed for 2000 iterations, for a total of 1.1 Billion simulated years per run. The high performance computing environment used to execute the scenarios is provided in Appendix F. The following section presents a description of the overall results, which include both the criticality parameters as well as maximum and minimum values of performance measures for each simulated scenario. Next is a description of outcomes for selected scenarios, where performance measures and reservoir elevations can be explored in further detail.

4.6.1 Overall results discussion

Using the output .npz files from the simulation, the criticality parameters and minimum (or maximum) performance measures were computed for each simulated scenario. The output files and output analysis code are formulated so that only the complete iterations (those where all scenario operating states both occurred and affected one another) for a given scenario are considered in the computation of that scenario's parameters. Results can be found in the electronic appendix, in the folder "Simulation_Results". The file "OutputResultsAll_base.xlsx" contains the simulation results for the base case, and "OutputResultsAll_dsi.xlsx" contains the results for the dam safety improved case. For the base case, there were a total of 9,669,654 failures simulated over a total of 857,102,076 completely implemented iterations (regardless of scenario), making the overall simulation flow control failure rate for the base case equal to 1.13%. This is not to be misinterpreted as the failure rate for the system – the failure rate for the system overall would be significantly reduced if the probabilities of occurrence of the operating states were taken into consideration. For the dam safety improved case, only 2 out of 809,563,591 complete iterations resulted in dam failure, making the overall simulation flow control failure rate for the system equal to 2.47×10^{-7}%. Again, this is not to be misinterpreted as the overall estimated failure rate for the dam system – it simply represents the proportion of simulations that resulted in dam failure, regardless of the probabilities of each simulated scenario. The number of failures observed in the dam safety improved case was extremely

small – likely as a result of the significantly larger free overflow spillway being capable of passing the largest inflows in the synthetic record if the reservoir elevation is below a certain level. Comparing the simulation failure rates for the two cases shows how significantly the increased overflow spillway capacity affects the rates of failure for the scenarios. These results are summarized in Table 4-11.

Investigating specific scenarios can give further insights into the vulnerability of the system to various combinations of events. Within the results spreadsheets (OutputResultsAll_base.xlsx and OutputResultsAll_dsi.xlsx), column headers are used to describe the parameters calculated for each scenario. In this analysis, the operating state impacts did not depend on the causal factors – that is, a single operating state would have the same range of impacts regardless of what causal factor resulted in it occurring. Because of this, it is possible to combine scenarios with the same sets of adverse operating states into a causal factor-independent set of scenarios, which have more observations and therefore an improved estimate of the criticality parameters. This sorting resulted in a total of 6,144 combined scenarios that can be easily analyzed in more detail. These are presented in "Results_CombinedComps_base.xlsx" and "Results_CombinedComps_dsi" for the base case and the dam safety improved case, respectively. Within each of these results spreadsheets, there are different tabs containing the complete results (All) as well as filtered results which contain scenarios that have the same number of adverse causal factors ($N = 1 \ldots 5$).

Sorting the failure rate values in the combined results spreadsheet (All) for the base case shows that for 229 scenarios, the failure rate was greater than or equal to 10%. Scrolling through this list shows that all of these scenarios involved a restriction in capacity as a result of the gate being failed, either closed or in place. Interestingly, another component that frequently appears in the most severe scenarios is the penstock rupture. This is a direct result of the outage length of penstock rupture scenarios – which can exceed a full year following the event. In this case, the reservoir would initially drain through the penstock until it is below the sill of the intake gate, which would then be closed. After the intake gate is closed, the power water passages are out of service for a significant amount of time. This means that while initially some uncontrolled release may be observed, there may be

longer-term complications associated with operating the reservoir once the power passages are isolated and the flow conveyance capacity has been lost. Turbine head cover failures are also higher up on the list for the same reasons. Other issues that come up within this more severe scenario list include site access and staffing issues, communications equipment (PLC/RTU) failures, and sensor issues.

Table 4-11: Overall results summary

	Base Case	**Dam Safety Improved**
Total number of years simulated (complete iterations)	857,102,076	809,563,591
Total number of failures simulated (complete iterations)	9,669,654	4
Simulation Failure Rate*	1.13%	2.47×10^{-7}%
Highest scenario failure rate	16.227%	0.005%
Average failure inflow threshold (mean 5-day inflow preceding failure)	114 m^3/s	835 m^3/s
Average failure inflow threshold (max 5-day inflow preceding failure)	160 m^3/s	1588 m^3/s

For the dam safety improved case, the two scenarios that lead to failure resulted from a loss in gate capacity (gate failing closed or in place), in combination with sensor issues, access delays. Additionally, both failure scenarios included a loss of power flow conveyance through either penstock rupture (and subsequent lengthy outage) or as a result of grid failure and resulting load rejection.

The overall average inflow thresholds – the minimum average/maximum daily inflow in the 5 days preceding failure – are 114 m^3/s and 160 m^3/s, respectively, for the base case. These increase to 835 m^3/s and 1588 m^3/s for the dam safety improved case (with only two

failures). Obviously, the failures in the dam safety improved case are a result of very high inflows that exceed the safe discharge capacity of system.

A few more general conclusions can be drawn from the overall results tables. The highest observed conditional failure rate in the base case was 16.227%, versus only 0.005% in the dam safety improved case. For both cases, dam overtopping failure and high reservoir elevations were most frequently occurring as a result of loss of flow conveyance capacity – specifically, losses in conveyance capacity involving the gate, which can pass significantly more flow than the power conduit. Sensor errors, communications failures and access/staffing issues were also significant contributors to overtopping failures and reservoir level excursions above the key levels.

Looking at the results based on the number of affected components may also provide useful insights into the most vulnerable aspects of the system. Filtering the list to only a single component being affected gives the results in Table 4-12 and Table 4-13, for the base case and the dam safety improved case, respectively. These tables have been abbreviated slightly (by reducing number of columns) to ensure the columns fit on the page. The columns in the table show the conditional failure frequency, failure inflow thresholds, conditional frequency of exceeding key reservoir elevations, the minimum discharge capacity and the maximum uncontrolled release. The final column shows the name of the affected component.

For the base case (Table 4-12), the obvious result is that the components whose failure results in the most significant capacity loss (the gate components causing the gate to fail closed or in place) lead to the greatest failure rates and highest reservoir levels. Next are the sensor errors, which can result in lack of conservatism in reservoir operations. None of the other components on their own lead to failure in the base case, but gate blockage, grid outages and sensor failures also caused reservoir elevations to exceed key levels. For the dam safety improved case, none of the components on their own lead to failure of the dam due to overtopping. Issues with communication or sensors lead to the highest reservoir levels. Interestingly, penstock ruptures and head cover failures also resulted in some scenarios with reservoir elevations exceeding key levels. This is an unexpected result that

may be a result of the operations planning algorithm not taking into account the lost power flow release capacity, and thus keeping the spillway gate closed during those scenarios. In future runs of this model, the unit availability function should be modified to reflect the lost ability to pass water through the power conduit following closure of the power intake gate.

Looking more closely at the reservoir level exceedances in the base case, it is clear that reservoir excursions above key levels were directly related to either a loss of capacity or loss of remote visibility (reservoir level sensor failure or error). For the base case, failure of the gate in the closed position had a 15% chance of resulting in overtopping of the earthfill dam (and a 1.6% chance of overtopping it enough to cause dam breach). The frequency of overtopping the earthfill dam was reduced to about 2.8% for the gate failing in place as a result of some residual discharge capacity resulting from the gate being stuck in the position it was at prior to failure. For the dam safety improved case, overtopping of the earthfill dam was avoided for all single affected component simulations, except for the penstock rupture which results in a lengthy outage that, as discussed above, may not (but should have been) be recognized by the operations planning algorithm. In general for the dam safety improved case, loss of visibility resulting from either sensor issues or communication system failure (PLCRTU) resulted in the most significant exceedances of key reservoir levels. Surprisingly, loss of conveyance through the gate alone was not enough to cause reservoir level excursions even resulting in spill, which is somewhat surprising. This is likely a direct result of the conservative operating strategy in the dam safety improved simulations, which target reduced reservoir elevations in the case of loss of gate functionality.

Looking at the minimum discharge capacity gives some context to why the reservoir elevations may have risen. For both cases, the most significant losses in flow conveyance capacity (the maximum active discharge capacity being 1655 m^3/s) resulted from gate issues – the gate failing closed, in place, or being blocked. Not surprisingly, these were associated with higher likelihoods of exceeding key reservoir levels in the base case (but not in the dam safety improved simulations as a result of more conservative operations).

Table 4-12: Results for a single affected component, base case

Conditional Failure Frequency (%)	5-day Inflow Threshold (average daily)	5-day Inflow Threshold (maximum daily)	Conditional Frequency of Exceeding El. 377.95 m (WL Max)	Conditional Frequency of Exceeding El. 378.41 m (free overflow spill)	Conditional Frequency of Exceeding El. 380.4 m (Concrete dam)	Conditional Frequency of Exceeding El. 381.42 m (Earthfill dam)	Minimum Discharge Capacity	Mean time to failure (days)	Components
1.63	147.54	229.20	41.41	40.31	36.41	15.53	65.00	34.36	Gate fails closed
0.53	195.65	242.17	18.20	17.37	14.08	2.84	69.91	45.00	Gate fails in place
0.05	753.10	1995.10	10.36	8.50	3.77	0.25	1655.00	7.00	Sensor Error
0.00	NA	NA	48.82	41.81	18.09	0.25	383.60	NA	Gate opening
0.00	NA	NA	26.82	22.02	4.37	0.00	1655.00	NA	Grid
0.00	NA	NA	0.02	0.00	0.00	0.00	1655.00	NA	Sensor Fail
0.00	NA	NA	0.00	0.00	0.00	0.00	1655.00	NA	PLCRTU
0.00	NA	NA	0.00	0.00	0.00	0.00	1590.00	NA	Penstock
0.00	NA	NA	0.00	0.00	0.00	0.00	1590.00	NA	Head Cover
0.00	NA	NA	0.00	0.00	0.00	0.00	1590.00	NA	Generator
0.00	NA	NA	0.00	0.00	0.00	0.00	1655.00	NA	Gate collapse

Table 4-13: Results for a single affected component, dam safety improved case

Conditional Failure Frequency (%)	5-day Inflow Threshold (average daily)	5-day Inflow Threshold (maximum daily)	Conditional Frequency of Exceeding El. 377.95 m (WL Max)	Conditional Frequency of Exceeding El. 378.41 m (free overflow spill)	Conditional Frequency of Exceeding El. 380.4 m (Concrete dam)	Conditional Frequency of Exceeding El. 381.42 m (Earthfill dam)	Minimum Discharge Capacity	Mean time to failure (days)	Components
0.00	NA	NA	51.55	43.56	0.00	0.00	1655.00	NA	PLCRTU
0.00	NA	NA	40.05	39.07	0.00	0.00	1655.00	NA	Sensor Fail
0.00	NA	NA	28.34	24.59	0.00	0.00	1655.00	NA	Sensor Error
0.00	NA	NA	18.17	17.11	0.03	0.00	1590.00	NA	Penstock
0.00	NA	NA	12.55	9.91	0.00	0.00	1655.00	NA	Grid
0.00	NA	NA	0.18	0.14	0.00	0.00	1590.00	NA	Head Cover
0.00	NA	NA	0.00	0.00	0.00	0.00	1590.00	NA	Generator
0.00	NA	NA	0.00	0.00	0.00	0.00	383.60	NA	Gate opening
0.00	NA	NA	0.00	0.00	0.00	0.00	69.91	NA	Gate fails in place
0.00	NA	NA	0.00	0.00	0.00	0.00	1655.00	NA	Gate collapse
0.00	NA	NA	0.00	0.00	0.00	0.00	65.00	NA	Gate fails closed

Sensor errors did not result in any losses in discharge capacity but lead to increased reservoir levels in both cases through improper reservoir level operation.

Another indicator of the criticality of a scenario is the mean time to failure. For the base case, this ranged from 34-45 days in the scenarios with a loss of conveyance through the gate. This reduces to only 7 days in the case of reservoir level sensor errors, which indicates the potential severity of operating the reservoir assuming the reservoir level is lower than it actually is.

For scenarios with two affected components, the results for the base case and dam safety improved case are shown in tab N=2 of "Results_CombinedComps_base.xlsx" and "Results_CombinedComps_dsi.xlsx", in the Simulation_Results folder of the electronic appendix. There are 77 total combinations of components in these two-component scenarios (these combinations represent the combined scenarios which take into account results from the same scenario with different causal factors). For the base case, the combinations with the highest failure frequencies (2-15%) involved failure of both gate and power discharge components. Despite failures of the turbine head cover and penstock resulting initially in uncontrolled releases, the long-term impacts of these component failures is lengthy outages of the discharge facilities (once the intake gate is closed) – this means lower overall flow conveyance capacity in the long term. These higher failure frequency cases resulted in a 19-35% chance of overtopping the earthfill dam – even if the overtopping did not lead to a failure, significant damage would be observed in these cases. For the dam safety improved case, the combined loss of both power and gate releases lead to some instances of overtopping of the concrete dam, which could potentially cause significant damage. Free overflow spill was observed more frequently when both gate and power flow release facilities were out of service, and sensor issues in combination with gate failures also had a high conditional frequency of free overflow spill.

In the N=3 tabs of the same spreadsheets, the three-component combined scenarios are presented. For the base case, similar results are seen where the scenarios resulting in a complete loss of controlled discharge capacity (both gated and power flow releases) had the highest failure rates. The most extreme case involved a penstock rupture and

subsequent outage of the power flow release facilities, the gate failing in place and in the closed position – which had a 16% failure rate and a 43% chance of overtopping the earthfill dam. In this case, there are two potentially overlapping conditions of the gate – failed in place and failed closed – in the simulation model, the gate failing closed takes precedence over failures in place. For the dam safety improved case, the scenarios with the most significant overflow frequencies tended to include gate outages or capacity restrictions, loss of flow through the power conduit, and either sensor errors and/or communications equipment failures.

Another important observation is that for a relatively small proportion of the scenarios simulated, there may not have been enough complete iterations simulated to develop a meaningful characterization of the scenario. This is because in the post-processing, an analysis is done that determines whether all of the events occurred and affected one another. For some scenarios, events may be initiated after the system has already recovered from preceding events. In this case, the iteration is not representative of the cumulative effects of the combination of events and is filtered out of the scenario results. This can be observed by sorting the "OutputResultsAll_base.xlsx" and "OutputResultsAll_dsi.xlsx" by the column "Number of simulation-years". In the base case, about 30,000 scenarios had less than 500 complete iterations out of 2000 simulated, and 1170 of these had less than 100 complete iterations. About 32,000 scenarios in the dam safety improved case had less than 500 complete iterations out of 2000 simulated years, and 519 of these had less than 100 complete iterations. This indicates a significant limitation of the modelling framework applied in the case study – the number of iterations completed may not provide sufficient data with which to estimate credible conditional failure or reservoir level exceedance frequencies. This observation indicates that additional computing time may be required to properly analyze scenarios without sufficient data points – perhaps by setting some minimum complete iteration threshold within the simulation. It is worth noting that these scenarios involved a higher number of events occurring. This means that the time frame within which the different events can occur is relatively large (since it is equal to the sum of the Monte Carlo generated outage lengths and can be up to 365 days). As such, there may be several instances where the events do not affect one another (the reservoir level

recovers prior to the next subsequent event). This observation will be an important consideration in future applications of this methodology.

4.6.2 Assessment of individual scenario outcomes

Another output from the simulation model is arrays containing the dynamic performance indicators for complete iterations of each scenario, as well as the reservoir levels. These can be plotted to visually represent the system behaviour in response to various input scenarios and Monte Carlo parameters. Five scenarios have been selected and plotted to illustrate how results individually can be compared between the two cases. The summary tables containing the key data for each of the selected scenarios are shown in Table 4-14 and Table 4-15 for the base case and the dam safety improved case, respectively.

The first scenario involves the gate failing in the closed position as a result of ice, the grid being failed due to wind, and the site access being delayed due to traffic issues (Seed number 301490). Figure 4-20 contains the plotted reservoir elevations (in the first row), flow conveyance capacities (second row) and total uncontrolled releases (third row), for the base case (first column) and the dam safety improved case (second column). For the reservoir levels, the mean value is shown in black and the 10^{th} and 90^{th} percentiles are shown in darker grey. Each light grey line represents the dynamic reservoir level response for a single iteration of the scenario. Only completely implemented iterations are plotted – that is, scenarios in which the dam failed, or the events did not affect one another are not included. The length of each light grey line depends on the length of time within which the reservoir differed from the “normal” reservoir elevations for the same time period and inflow. For this scenario, the maximum length of time for which the reservoir deviated from the normal elevation was 250 days.

Looking at the reservoir elevation plots, it is immediately clear that the reservoir elevations in the base case were significantly higher than in the dam safety improved case. In the base case, no significant efforts are made to operate the system more conservatively given a loss in capacity. In contrast, for the dam safety improved case, the target reservoir elevation is

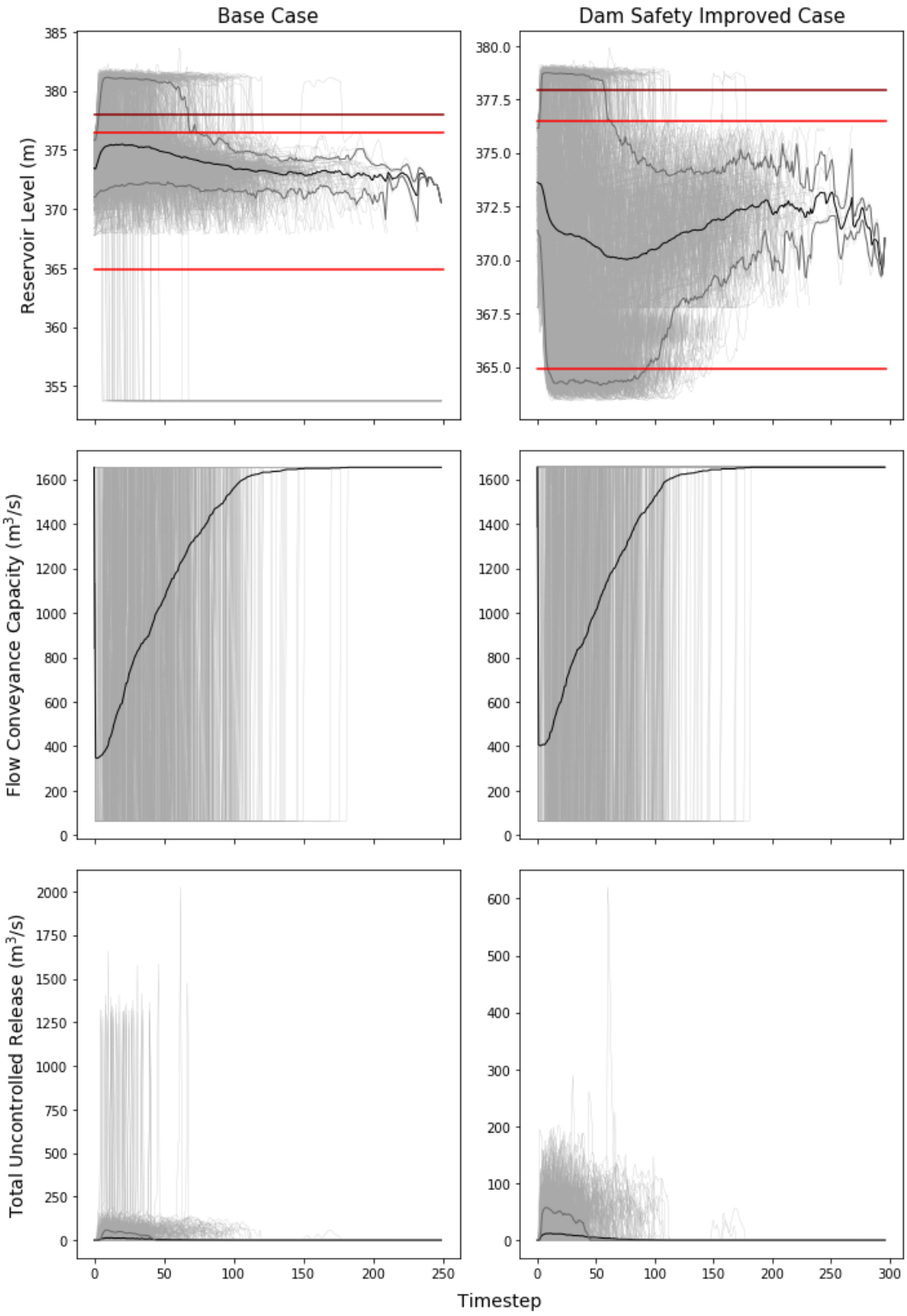


Figure 4-20: Dynamic results for seed 301490

reduced when the gate becomes unavailable. The reservoir is then lowered with whatever capacity is available, creating more storage should inflows exceed the remaining available capacity. As such, the dam safety improved case 90th percentile reservoir levels rise to a maximum of El. 379 m, which is significantly less than the El. 381 m observed in the base case. Failures within the base case are observed when the reservoir level sharply drops to El. 353.75 m. There are a total of 44 failures observed in the base case, with a failure rate of around 4%. The mean time to failure in this scenario was about 24 days. There were no failures observed in the dam safety improved case. It is also worth noting that the reservoir elevations dropped below the normal minimum (NMin) in the dam safety improved case. It is not immediately clear why this is the case since the target reservoir elevation is equal to NMin. The problem results from the operations planning algorithm not accounting for any free overflow spill when the projected reservoir elevations exceed the sill of the overflow spillway – in these cases, the reservoir level is reduced more than is necessary to achieve the NMin target. Future runs of the model should address this issue.

The total active flow conveyance capacities are plotted in the second row of Figure 4-20. The black line represents the mean values. The results are similar for both the base case and the dam safety improved case. One issue with these values is that the grid failure does not register as a loss of capacity though the power conveyance system, despite resulting in a load rejection and closure of the wicket gates. This component interaction is programmed into the simulation model, but not accounted for in the calculation of available capacity. Again, future runs of the model can be modified to address this problem. Because of this, the minimum flow conveyance capacity recorded for both scenarios was 65 m^3/s, which is the maximum flow that can be passed through the power conduit.

The third row shows the uncontrolled releases for the system, which are clearly significantly higher in the base case as the concrete and earthfill dams are overtopped. The maximum uncontrolled release for the base case was around 2000 m^3/s, and about 620 m^3/s in the dam safety improved case. The average uncontrolled release was similar for both the base case and the dam safety improved case.

It is also possible from the dynamic reservoir elevation plots to determine the conditional reservoir level exceedance frequencies for the scenario – that is, the percentage of time where the observed reservoir elevations for the scenario exceeded various levels. The daily reservoir level values are recorded from each complete iteration (where all events occurred and affected one another) and the percent of observations exceeding various reservoir levels is calculated. Figure 4-21 contains the conditional reservoir level exceedance frequencies for seed 301490, with the base case shown in red and the dam safety improved case shown in blue. This graphic is an excellent indicator of the improvement made by the dam safety improved case over the base case. The difference between the two lines is indicative of the level of improvement gained by the system upgrades and operating strategies employed in the dam safety improved case.

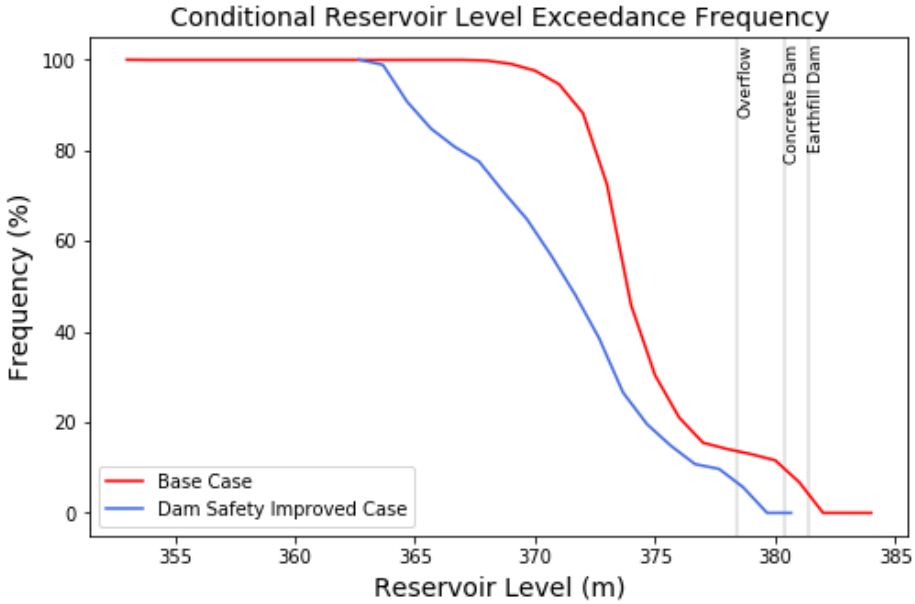


Figure 4-21: Conditional reservoir level exceedance frequencies for seed 301490

The second scenario involves debris blockage of the gate as well as a reservoir level sensor error resulting from temperature fluctuations causing instrument decalibration (seed 386196). The results are shown in Figure 4-22, where the first row shows the reservoir levels with the base case in the first column and the dam safety improved case in the second column. For this scenario, results show similar mean and 90th percentile reservoir elevations, with higher maximum levels observed in the base case. The higher maximum

levels are a direct result of the increase in free overflow spill capacity, which helps to offset the loss in capacity caused by debris buildup at the gates. There are some excursions below the normal minimum (NMin) reservoir level in both cases as a result of the sensor errors. The dam failed by overtopping in two scenarios for the base case and the average time to failure was 146 days.

The second row shows the active flow conveyance capacities for the base case and dam safety improved case in the first and second columns, respectively. The results are similar for both cases, with average values that are almost equal. The debris blockage is predetermined using the Monte Carlo randomization of scenario input parameters, however the length of time for which the debris blockage remains depends on the system inflows. When inflows fall below 65 m^3/s, the simulation model assumes that debris can be removed from the gate and capacity is restored.

In the third row, uncontrolled releases are presented for the base case and the dam safety improved case in the first and second column. Again, uncontrolled releases involve any free overflow spill, as well as dam breach flows and flows from penstock rupture or gate collapse. In this case, the majority of observed uncontrolled release is due to overflow spill, which may be through the overflow spillway but potentially can include dangerous concrete and earthfill dam overtopping. In the base case, there are two spikes when the uncontrolled releases exceed 1250 m^3/s, at approximately day 120 and day 180. These correspond to the iterations where dam breach occurred. Omitting these two scenarios, the overall uncontrolled release observed in the dam safety improved case was slightly higher, likely as a result of the increased free overflow spillway capacity at lower elevations.

Figure 4-23 contains the conditional reservoir level exceedance plots for seed 386196. The plots for both the base case and the dam safety improved case are very similar, with the only notable difference at the tail end of the curve where the maximum observed elevations in the base case exceeded those observed in the dam safety improved case. This small difference can be attributed primarily to the increased free overflow spillway capacity in the dam safety improved case – the decrease in the exceedance line occurs just above the level at which free overflow spill is initiated.

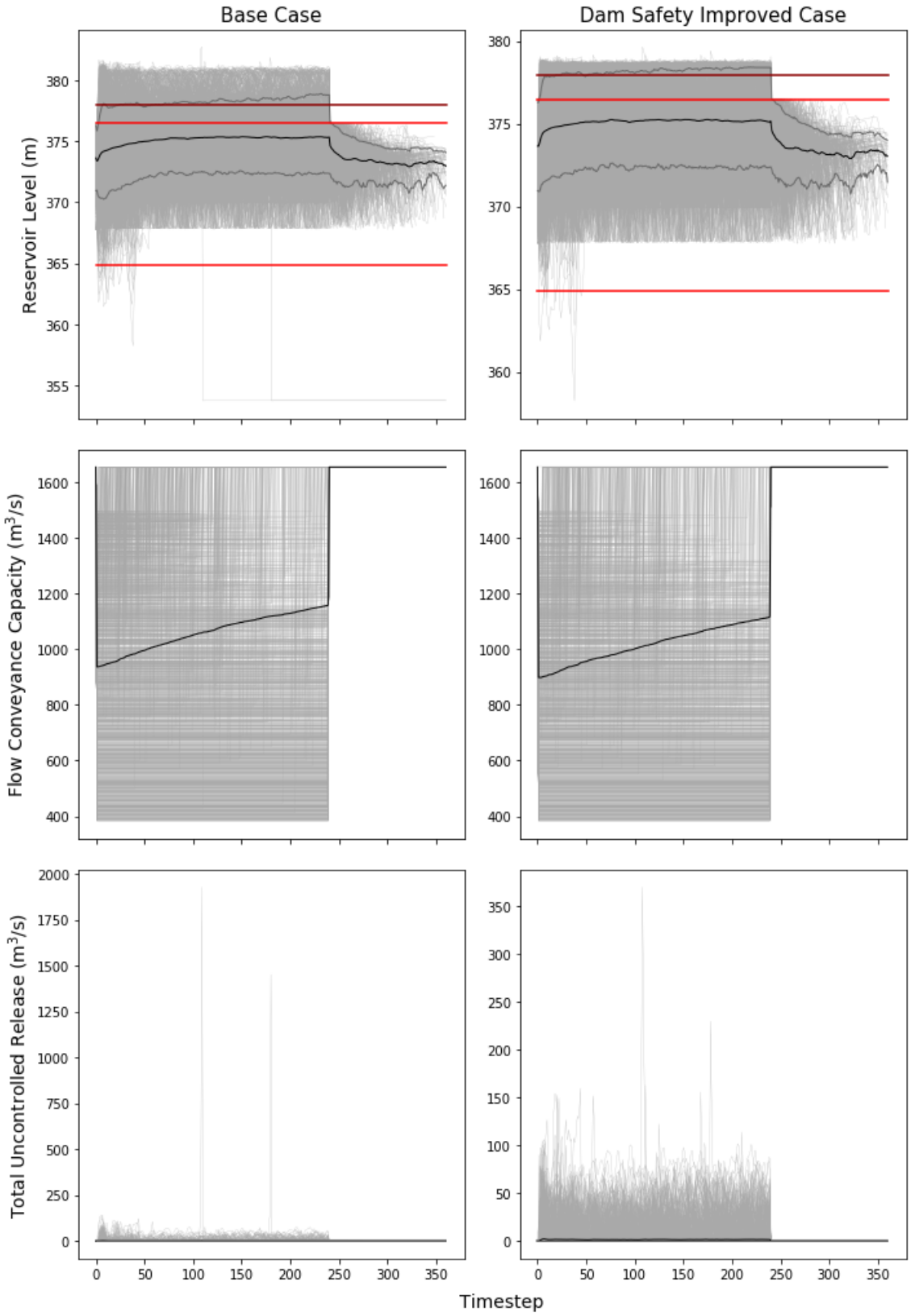


Figure 4-22: Dynamic results for seed 386196

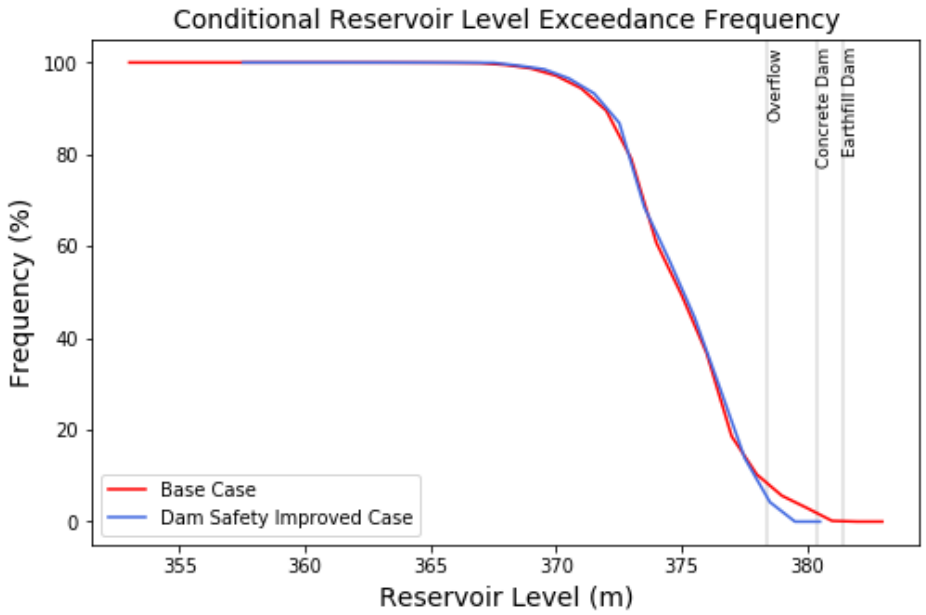


Figure 4-23: Conditional reservoir level exceedance frequencies for seed 386196

The next scenario is one of the more extreme combinations of events that lead to the highest combined scenario failure rate in the base case. This scenario involves failure of the gate in place, failure of the gate closed and a penstock rupture (seed 403429). For the reservoir elevations, a clear improvement is seen in the dam safety improved case in comparison with the base case. The reservoir level 90th percentile is around El. 379 m for the dam safety improved case, and El. 381.13 m in the base case. Because of the higher maximum reservoir elevations in the base case, a large number of failures are observed (296) and the failure frequency is quite high (17.3%). The average time to failure in the base case was 114 days. There are no failures in the dam safety improved case, partly due to the increased free overflow spillway capacity, and partly due to the operator reducing the reservoir level and operating more conservatively (if reservoir drawdown capacity is available).

The second row shows the available active flow conveyance capacity of the system. For both cases, the available capacity drops to zero when both power and gated releases are unavailable as a result of the penstock rupture. The 65 m^3/s capacity of the power conduit is seen at the top part of the figure where the gate capacity recovers but the penstock is still unavailable. The results are similar between the dam safety improved and the base case.

The third row shows the total uncontrolled releases for the base case and the dam safety improved case in the first and second column, respectively. The total uncontrolled releases include penstock rupture flows, free overflow, dam overtopping flows and dam breach flows, so it can be somewhat difficult to decipher what the contributing factors are in a scenario which could have all of these. Obviously, the dam breach flows significantly increase the maximum values observed in the base case. The uncontrolled releases in the dam safety improved case have a maximum of about 390 m^3/s with the 90^{th} percentile being around 80 m^3/s. For the base case, the 90^{th} percentile values are around the same. The initial spike in the mean uncontrolled release values can be attributed to penstock ruptures, which may happen on day 1 of the simulation for about 1/3 of the simulated scenarios (based on the random selection of initiating event). In the base case, the initial spike is near 100 m3/s, and this is reduced to about 15 m^3/s in the dam safety improved case since the intake gate can be closed under rupture flows.

Figure 4-25 contains the conditional reservoir level exceedance frequencies for seed 403429. There is a relatively close agreement between the base case and the dam safety improved case, with the latter actually exceeding the base case values for elevations less than El. 378.5 m. This is somewhat surprising given the differences observed in the dynamic reservoir elevation plots for the same scenario. One potential contributing factor is that less water is released from uncontrolled penstock rupture flows in the dam safety improved case (since the intake gate closes under rupture flows). This means the reservoir level may be higher when gate failures initiate, or that the reservoir level does not decrease by a substantial amount if the penstock failure is initiated after the gate failure. The result is moderately higher reservoir elevations through parts of the curve up until free overflow spill is initiated (El. 378.41 m). Above El 378.5, the dam safety improved case drops off below the base case curve, meaning the reservoir level did not reach the same maximum levels. This is a result of the increased free overflow spillway capacity, which helps maintain reservoir levels below El. 380 m.

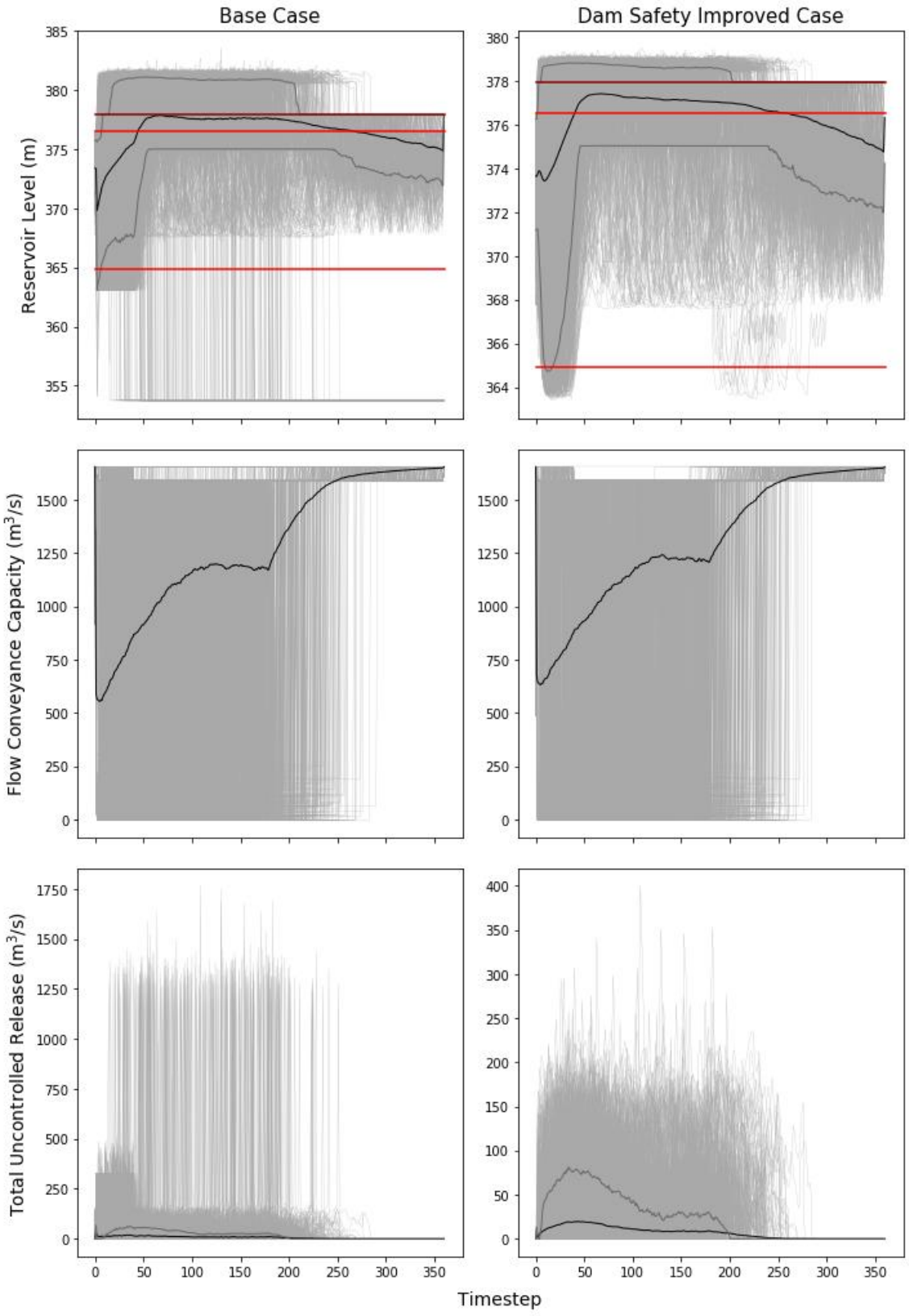


Figure 4-24: Dynamic results for seed 403429

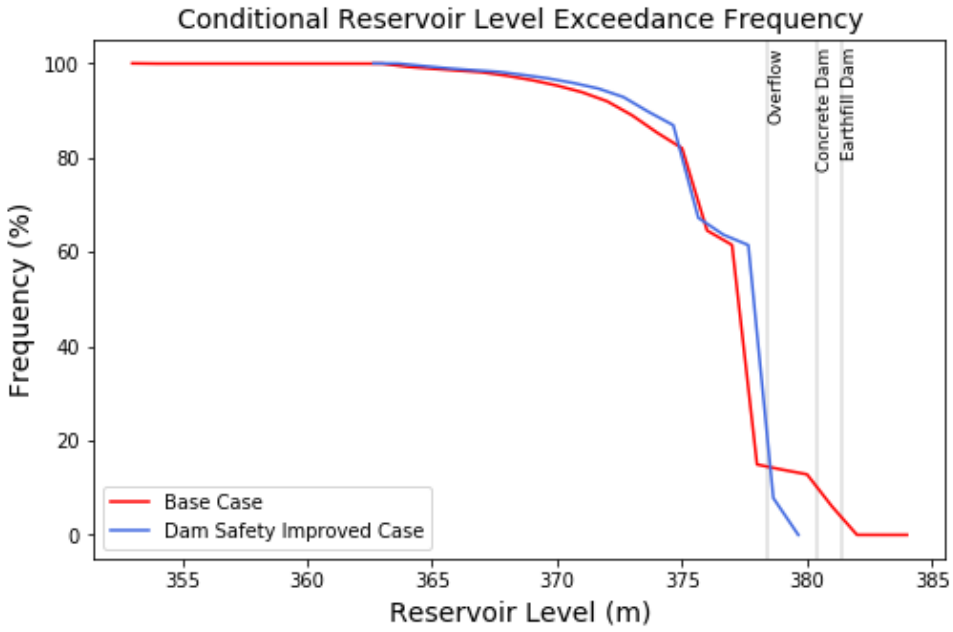


Figure 4-25: Conditional reservoir level exceedance frequencies for seed 403429

The next scenario involves a single component failure – a penstock rupture due to an earthquake (seed 403440). The reservoir levels are shown in the first row, with the base case in the first column and the dam safety improved case in the second column. The key difference between the two plots is that the reservoir level drops significantly lower in the base case. This is a direct result of the ability of the intake gate to close under rupture flows in the dam safety improved case. This results in significantly smaller uncontrolled release flows (see the figures in the third row). It is important to note that the flows recorded represent the average daily flows, and that peak outflows may be significantly higher for the dam safety improved case where the intake gate closes within an hour of rupture. The loss in capacity observed is related to the inability to pass flows through the generating unit while the penstock is being repaired.

Figure 4-27 contains the conditional reservoir level exceedance frequencies for the base case (red) and the dam safety improved case (blue), respectively. The dam safety improved case has a higher conditional reservoir level exceedance frequency in comparison with the

base case, as a result of a smaller volume of water being lost through the penstock. This means the reservoir remains at a higher elevation throughout the course of the scenario.

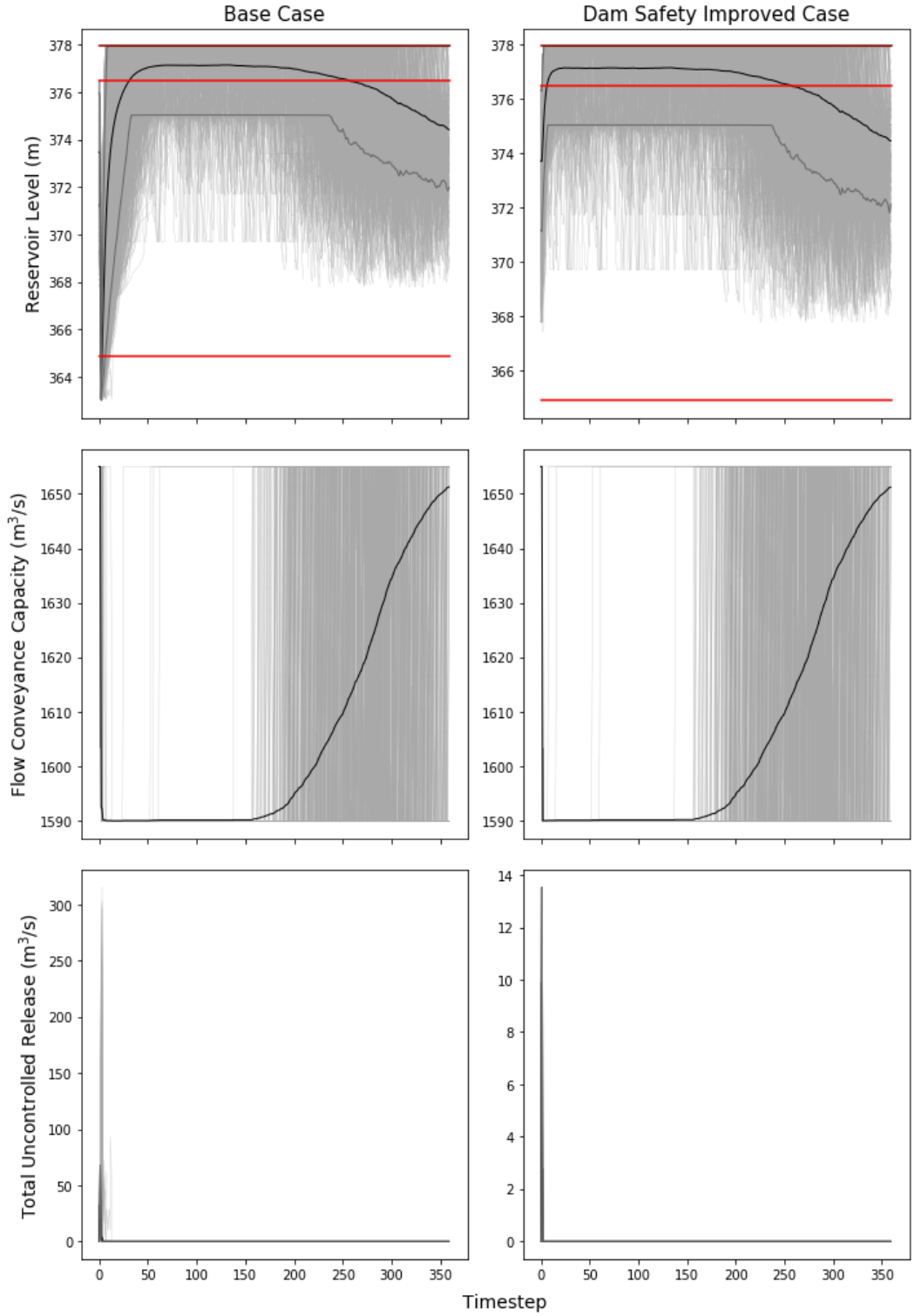


Figure 4-26: Dynamic results for seed 403440

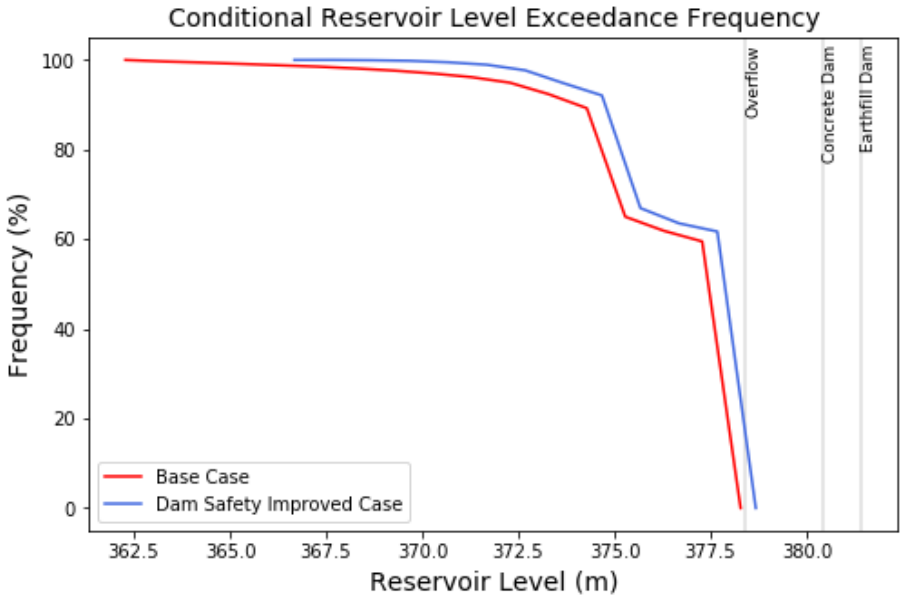


Figure 4-27: Conditional reservoir level exceedance frequency, seed 403440

The final scenario selected for discussion is one of only two scenarios that resulted in an overtopping failure in the dam safety improved case. This scenario involved a number of events: the gate failing closed, a sensor error, delays in accessing the site and an outage of the grid (seed 281617). Figure 4-28 contains the dynamic results for this scenario. The first row of the figure shows the dynamic reservoir levels for the scenario. In the base case (first column) many of the scenarios exceeded the water licensed maximum level and 21 dam breaches occurred. The average time to failure in the base case was about 47 days. In the dam safety improved case, there were excursions above the water licensed normal level, however these tended to be less extreme than in the base case. This is in part due to a larger free overflow spillway, and also because of the operating strategy to reduce the reservoir elevation during outages affecting the gate. One failure is observed, occurring within three days of the start of the scenario. In both cases, the capacity loss is similar, dropping down to the turbine only being available during the gate outage. Uncontrolled releases are generally similar between the two cases, with the more extreme spikes in the base case corresponding with the dam failures. Figure 4-29 contains the reservoir level exceedance

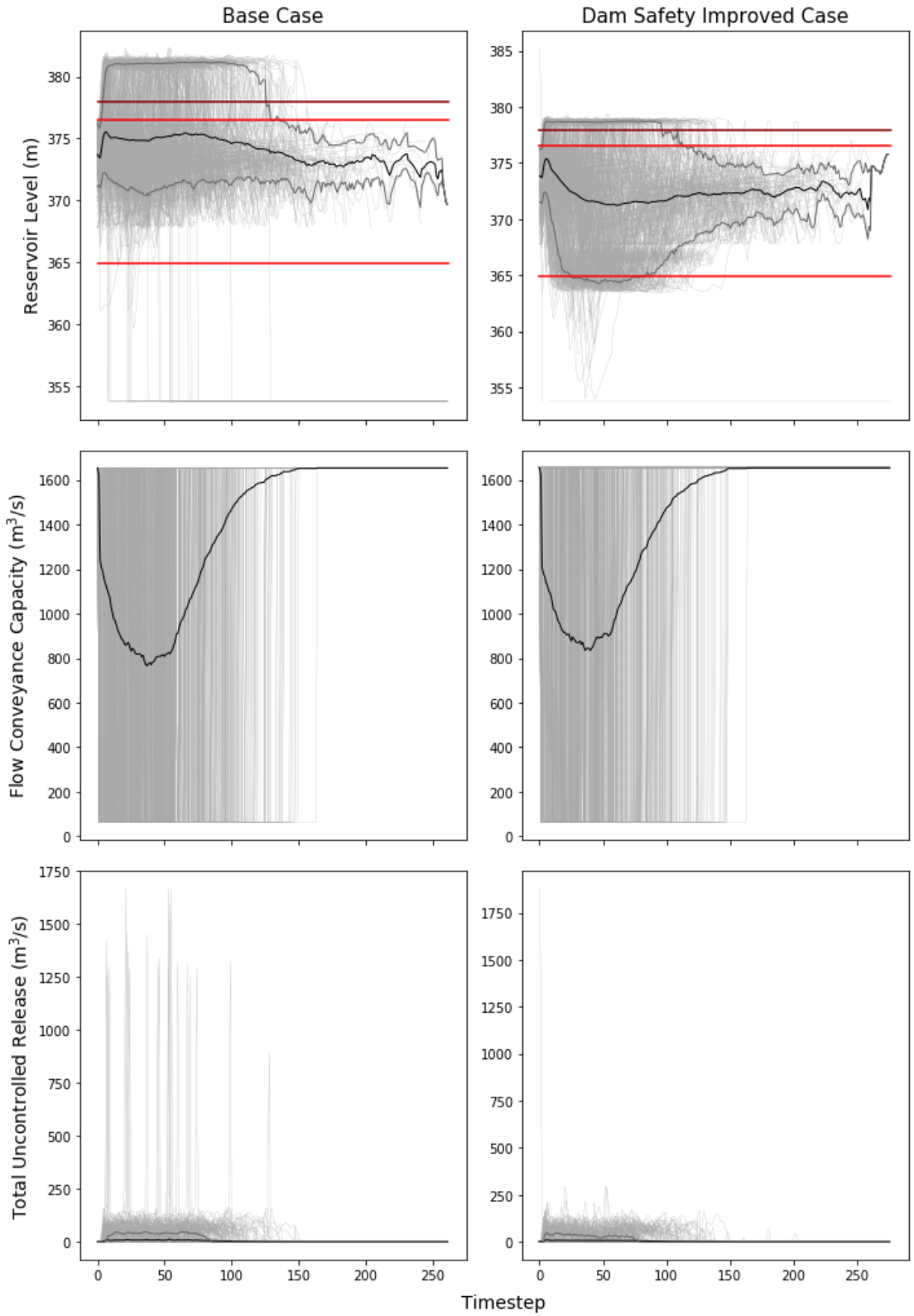


Figure 4-28: Dynamic results for seed 281617

plots for seed 281617. There is a significant difference between the two curves, with the dam safety improved case generally spending less time at higher reservoir elevations than the base case.

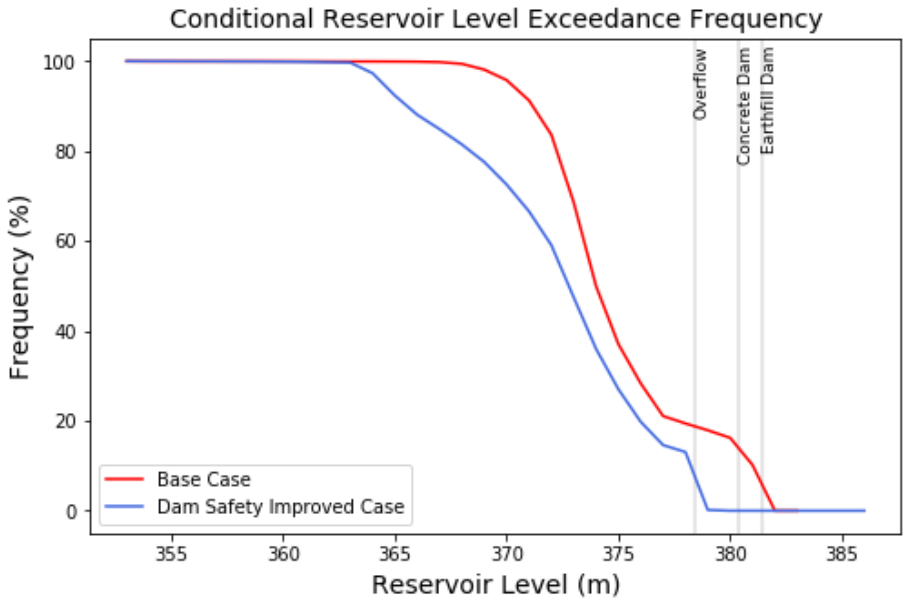


Figure 4-29: Conditional reservoir level exceedance frequency, seed 281617

Overall, the results from these individual example scenarios provide useful information that can help to better understand the dynamic system response to individual scenarios. Comparing the results between the two cases gives a good indication about the improvements made by introducing refined operational strategies and improving infrastructure. The conditional reservoir level exceedance frequencies provide an additional indication of whether there are significant improvements between scenarios.

The summary of the results from each of the highlighted scenarios can be found in Table 4-14 and Table 4-15, respectively. These tables, along with the dynamic results and reservoir time exceedance plots provide a good comparison between the two runs of the model. Ultimately, there are a large number of scenarios to be discussed and only a very small subset were analyzed in this thesis. However, the analysis of these scenarios provides some indication of how the modifications to the system improve the performance in these

extreme conditions. The dynamic analyses, as well as the tabular outputs from the simulation (discussed in the previous section) are useful outputs that can help identify vulnerable components of the system. Comparing the results between the different model runs can help build a business case for upgrades to the system and may be helpful to guide emergency planning activities.

Table 4-14: Summary of results from individual scenario outcomes, base case

Seed Number	Conditional Failure Frequency (%)	5-day Inflow Threshold (avg daily)	5-day Inflow Threshold (max daily)	Minimum Discharge Capacity (m3/s)	Maximum Uncontrolled Release (m3/s)	Maximum Reservoir Level (m)	Number of Simulation-years	Average time to failure (days)	Components
301490	4.37	150	164	65	2026	383.65	1007	23.91	Access delay, grid failure, gate fails closed
386196	0.10	350	582	383	1928	382.74	1992	145.50	Sensor error, gate opening blocked
403429	17.32	109	164	0	1766	383.61	1709	114.59	Penstock rupture, gate fails in place, gate fails closed
403440	0.00	NA	NA	1590	316	377.94	2000	NA	Penstock rupture
281617	3.11	154	168	65	1671.01	382.33	676	46.76	Access delay, sensor error, grid failure, gate fails closed

Table 4-15: Summary of results from individual scenario outcomes, dam safety improved case

Seed Number	Conditional Failure Frequency (%)	5-day Inflow Threshold (avg daily)	5-day Inflow Threshold (max daily)	Minimum Discharge Capacity (m3/s)	Maximum Uncontrolled Release (m3/s)	Maximum Reservoir Level (m)	Number of Simulation-years	Average time to failure (days)	Components
301490	0	NA	NA	65	620	379.92	NA	917	Access delay, grid failure, gate fails closed
386196	0	NA	NA	383	370	379.65	NA	1830	Sensor error, gate opening blocked
403429	0	NA	NA	0	400	379.52	NA	1691	Penstock rupture, gate fails in place, gate fails closed
403440	0	NA	NA	1590	14	377.94	NA	2000	Penstock rupture
281617	0.17	835.35	1588.17	65	1882.25	385.29	3	605	Access delay, sensor error, grid failure, gate fails closed

4.7 Summary

The methodology developed in this research was applied to the Cheakamus System, located near Squamish, BC. First, a detailed representation of the Cheakamus System was created within the components operating state database. Operating states, impacts and causal factors were defined in the operating states database. The combinatorial procedure developed in this research was applied to the outputs of the dataset. For the detailed representation of Cheakamus, a total of 1.83×10^{27} operating state combinations (scenarios) were defined. This is a good indication of the dimensionality of the problem – the number of potential scenarios increases exponentially with the level of detail.

A simplified proof-of-concept representation of Cheakamus was developed next, with a single gate and a single turbine. This representation of the system returned 552,960 potential scenarios. A system dynamics simulation model representative of the simplified system was developed and tested by comparing the results with historical operations data. The simulation model was run 2000 times for each of the 552,960 scenarios, and for two separate cases (a total of 2.2 Billion years of inflows were run through the simulation model). The base case is representative of the simplified Cheakamus System with a smaller free overflow spillway. The dam safety improved case represents the same system with a free overflow spillway size that mimics the real system. The dam safety improved case also included a number of operational improvements, including power flow intake gates that could be closed under extreme flows, as well as improved communications redundancy and more conservative operating rules that aim to prevent reservoir level excursions above target levels. Each of the 2000 iterations for a scenario contained unique Monte Carlo-varied parameters for timing, impact magnitude and inflows. Synthetic inflows outside of the historically observed range were simulated using a stochastic weather generator and a hydrological model. The Monte Carlo variation of inflows was done by randomly choosing a day and year from the historical record and sampling the subsequent inflows. A triangularly distributed variable was generated using the minimum, maximum and average specified impact magnitude from the database for each adverse operating state simulated. Timing of adverse operating states was done by shuffling the operating states and assigning random times from within the first six months of the year long simulation.

In terms of implementation, high-performance computing (HPC) resources are required for implementation of this ambitious simulation exercise. In this research, there were 1.11 Billion years of daily simulations performed, with 2000 Monte-Carlo iterations for each of the 552,960 Million scenarios. This was performed twice – once for a base case and once for a dam safety improved case. The total of 2.22 Billion simulation-years was made possible by development of a very efficient simulation model and use of a serial farming approach that runs many simulations in parallel on HPC clusters. The simulations were completed in a period of about three weeks, though results would vary depending on the resources available and the HPC clusters utilized. The speed with which this large simulation task was completed is considered to be a substantive achievement.

The results from the simulation were analyzed by sorting and filtering the lists of results for each scenario. Scenarios with 1, 2 and 3 contributing components were filtered out and discussed to gain insights about the most critical components that could contribute to failure. The dam failures in the base case occurred in 1.3% of the total simulated years. For the dam safety improved case, this was reduced to 2.47×10^{-7}% of simulated years. These failure rates are not to be confused with estimates of overtopping failure frequency for the system as a whole – in order to compute that, operating state frequencies must be pre-defined. The proportion of failures simulated does give some indication as to the level of improvement made when the dam safety improved modifications are made to the system. For the base case, loss of conveyance through both power and gate release facilities was the most significant contributor to failure. For the dam safety improved case, only two failures were observed – both of these corresponded with a loss of gate and power flow capacity, sensor issues and the most extreme flood in the synthetic record. This is indicative of a much more robust overflow spillway system in the dam safety improved version of the model.

An assessment of some of the individual scenario results was also provided. Five scenarios were selected and the dynamic reservoir elevations and performance measures were plotted, along with the reservoir exceedance frequencies. These plots provided useful indications of the difference between the base case and the dam safety improved case for a given scenario, and could be used to better understand the system response to the scenarios.

Because the case study and analyses in this work were representative of a very simplified version of the Cheakamus System, they should not be interpreted as conclusions for the existing Cheakamus Project. Comparing the base case results (with a smaller free overflow spillway) to the dam safety results (with a free overflow spillway equal in size to the Cheakamus Dam), showed that the free overflow spillway was, in all but two cases, able to prevent reservoir elevations from reaching the level assumed to fail the dam. The comparison of results between the two cases simulated shows how the methodology may be useful in quantifying the improvements made by various system upgrades and configurations. Results from the case study illustrated that the approach presented here could be useful to assist dam safety emergency response decision making, by indicating how critical a scenario is and roughly how long there is to regain control over the reservoir. In addition, the results may be useful in operational decision making with respect to outages and operating rules, and could help build a business case for capital improvements to the system. The analysis was also useful in predicting potential combinations of event that could lead to failure, and identifying the events (or component states) that were most likely to result in significant safety impacts.

5 Discussion and Conclusions

Dam systems are arrangements of physical and nonphysical components which act to store and convey water for beneficial purposes such as power production, irrigation, water supply and flood control. Dams can be thought of as open systems, where inflows, outflows and disturbances cross the system boundary. Within the boundary of the system, feedbacks act to monitor reservoir levels and inflows, and adjust controls to maintain reservoir levels within target values, and meet desired outflow requirements, if possible. There are a wide range of potential constraints which may impact the ability of the dam to achieve its desired purpose for safe containment and conveyance of flows. The major research contributions in this work are (a) the systematic definition of combinations of events which can influence the ability to safely control flow in a dam system, and (b) the dynamic characterization of the system performance in response to these events using a Deterministic Monte Carlo simulation framework with a system dynamics simulation model.

The following paragraphs discuss the outcomes of this work as they pertain to the objectives outlined in Chapter 1.

- The first objective was to investigate the use of systems analysis and risk assessment concepts from within and outside of the dams industry in terms of their ability to determine potential operating scenarios for dam systems and the impacts scenarios have on system outcomes. This was achieved by looking at the relevant literature and evaluating the various techniques with respect to their ability to achieve the research requirements.

- The second objective was to develop an approach that helps define a more complete range of potential operating scenarios (operating constraints) than is possible using existing techniques alone. This was achieved through the use of a components operating states database that details each component, their operating states, operating state impacts and causal factors. Combinatorics was used to automatically convert the database entries into an exhaustive list of potential operating scenarios. The existing methodologies described in the literature review rely on expert judgement to determine possible combinations of events – of which

there are so many that it would be unreasonable for a team of experts to conceive of them. Thus, the number of scenarios that can be defined using this methodology far exceeds the scope of existing methods.

- The third objective was to develop an improved dam safety analysis methodology that facilitates systematic investigation of all potential operating scenarios and allows for the outcomes of individual scenarios to be characterized. The Deterministic Monte Carlo simulation framework proposed in this research is able to dynamically evaluate each possible scenario through a number of iterations. Scenarios are used as an input to the model, to ensure each scenario is systematically characterized and an equal amount of simulation effort is spent on each. This allows for a more complete assessment of potentially hazardous outcomes than has been achieved using existing techniques.

- The fourth objective was to develop a simulation approach that can handle complexity in system structure, feedbacks, interactivity and nonlinear behaviour and uses object-oriented modelling to improve model accessibility. The system dynamics simulation model developed in this work is well suited to this objective and can be built to as much detail as is required to adequately capture the failure modes of interest to the modeller.

- The fifth objective was to investigate dynamic indicators of system performance with respect to safety, as well as scenario criticality parameters that can be used to rank the importance of various scenarios from the simulation outcomes. A number of criticality parameters are proposed in this work, and these as well as the dynamic outcomes of the system performance were shown in the simulation results.

The following section contains a more detailed evaluation of the methodology which pertains to the requirements stated in Chapter 3. Next is a discussion of potential areas for future work.

5.1 Methodology evaluation

Like all approaches, the proposed methodology described in this research does have some limitations. A discussion of its strengths and weaknesses of the methodology is provided here. The requirements for a new approach to dam safety analysis were outlined in Chapter 3 and are repeated here with a more detailed discussion regarding the progress made towards each.

The first requirement is for an approach with reduced subjectivity. The approach presented in this research achieves this requirement. The automated generation of scenarios helps eliminate reliance on heuristic thinking and expert judgement with respect to combinations of possible system states. Use of STPA in the development of the operating states database for actively controlled system components such as gates and turbines also helps to reduce subjectivity. Despite the improvements in limiting subjectivity, there is still (and will always be) a requirement for expert judgement in the component operating state database population and the level of detail in which to model the system.

The second requirement of the approach developed is to address non-failure related constraints on system operation. All failure-based approaches are inherently limited in terms of the analysis of non-failures, and this is documented well by Leveson (2011) and Thomas (2012). Approaches which are focused on failures alone may miss a sub-set of potentially unsafe scenarios triggered by conditions that did not result from a failure. The proposed methodology is capable of assessing many of the scenarios not triggered by a failure. The database developed in this research includes both failure and non-failure operating states. The database and simulation model are well suited to deal with errors and delays which do not necessarily involve failure of components. There are, however, some non-failure related scenarios that were revealed through STPA analysis that were not captured within the simulation model presented in this work. Human factors and software requirements are issues that simulation is well-suited to address, so it may be possible in the future to improve the capabilities of the simulation model in this respect. Design flaws may be simulated, but need to be well understood and programmed into the simulation model. The simulations run in the case study do not specifically address these issues, and

the STPA portion of the case study (Section 4.1) highlights some of the scenarios that were missed within this application, but could potentially be represented in the future.

The next requirements on the list are to determine the potential constraints on system operation and to systematically determine potential combinations of these. The approach presented in this work achieves this requirement through the use of the operating states database population, and automatically generates combinations of operating states using the combinatorial procedure. Population of the database will benefit from the strengths of the systematic FMEA and STPA approaches.

Another requirement is to determine the likelihood of operational constraints (operating states) without significant simplifying assumptions. This remains a significant issue in probabilistic risk assessment that is difficult to address in the absence of supporting failure rate data. The methodology presented in this research does not attempt to address this problem. However, this research does determine the conditional probability of failure/reservoir level exceedance, given a scenario. Using this information, it may be possible to perform sensitivity analyses to assumptions regarding component probability of failure, without significant simulation effort. This represents one advantage over completely stochastic simulation models, which require re-simulation to analyze the sensitivity to assumed probabilities. The extension of this work to include full probabilistic assessment was not considered, since failure rate data for the components modelled was not available to provide a meaningful assessment.

The next few requirements are (a) quantifying the dynamic system response to operating scenarios, (b) including feedbacks and nonlinear behaviour, (c) capability to handle complexity. These are all dealt with specifically using the system dynamics simulation approach. System dynamics simulation is well suited to modelling the complex web of component interactions and feedbacks using object-oriented programming which is relatively transparent (interactions shown using stock and flow diagrams) and also easily modifiable. Inspection of the system structure is a useful way of gaining confidence in the model. The simulation model characterizes how the values of variables change with time – a direct output of the model is the reservoir level response to a particular scenario, which

is of significant importance for dam safety analysis. The outputs of the simulation may indicate emergent behaviour that results from component interactions. One major consideration for the simulation model presented is that the choice of simulation timestep must be selected such that all failure modes being considered are properly modelled. Penstock pressure transients and cavitation in turbines or spillway chutes happens over seconds, milliseconds or even shorter time intervals. These issues were not explored within this thesis, but could potentially be included in future applications of the work using nested processes within a simulation model that operates at a larger timestep (this would reduce the computational effort associated with such a fine time-resolution). While the approach presented is, in theory, capable of modelling the system with any desired level of complexity and at any time-resolution, there may be significant computational challenges when applying the methodology to very complex systems. As the complexity of the system being modelled increases, so does computational effort and number of scenarios to be analyzed. Future research should focus on improving the computational efficiency of the simulation model framework. Another time-related limitation is that the randomly initiated failures of components may not coincide with likely instances of failure in the real world. For example, in a real dam system, a spillway gate may be dormant or not "on demand" for a substantial period during the year. Failing the gate randomly may under-estimate the potential impacts this failure could have by initiating it when inflows are normal or low. That said, with regular gate testing being implemented in many dam safety programs across the world, it may not be unrealistic to detect a failed state during a low-flow period.

The next requirement is to assess the uncertainty in scenario outcomes. This is a particularly challenging issue in all modelling exercises. The uncertainty of scenario outcomes can be assessed by looking at the range of results from the Monte Carlo iterations of each scenario. By varying the simulation parameters and event timing, the sensitivity of the results to various inflows, event timing and event impact magnitudes is performed. There are a number of uncertainties in other model assumptions that have not been analyzed in this work and remain an important area for future work.

The ability to deal with common cause failures is another requirement of the methodology presented in this work. This is addressed within the operating states database and

simulation. Operating states which have the same causal factor are programmed to occur at the same point in time in the simulation. Computing the scenario probability (if the failure rate data are available) may be slightly more challenging, since care must be taken to ensure the probabilities of the common causes are not double counted.

The last two requirements of this research are the ability to calculate the conditional probability of failure for a given scenario, and the ability to calculate the probability of failure for the system as a whole. A direct outcome of the Monte Carlo simulation for each scenario is the conditional frequency of failure for that scenario, given the inflows and the range of impact parameters simulated. One limitation is that the number of data points with which to estimate these frequencies may be limited, since there may be many iterations within a scenario that are not representative of a "complete scenario" where all events are both occurring and impacting one another. These incomplete iterations, which are not representative of the scenario, increase the simulation effort without improving the result and result in fewer data points with which to calculate the overall failure likelihood. In the future, this could be dealt with by setting a minimum number of "completely implemented" iterations, or experimenting with the maximum timestep before which all events must occur within the scenario. Estimation of a system's overall frequency of overtopping failure is not a direct outcome of this research. However, assuming the frequency of each operating state can be estimated, it may be possible to perform a complete probabilistic analysis of simulation results. Ensuring calculations are correct may be challenging for common-cause failures, though there is some guidance in the literature on this subject. Running the simulation using Deterministic Monte Carlo will also facilitate a relatively straightforward sensitivity analysis to assumed operating state frequencies. This is an important area for future work.

In general, the approach presented in this work provides some key advantages over the existing techniques used within and outside of the dams industry. Traditional assessments tend to rely on techniques developed for use in industries that face different challenges than are experienced in dams systems. Dam systems are dynamic systems of many interacting components acting to control (both actively and passively) a randomly varying natural input (inflow). Determining the reservoir level response to various inflows and operating

constraints is not easily done using traditional failure modes brainstorming exercises or chain-of-event style analysis. Chain-of-events analyses (FTA and ETA) are limited in their ability to address interactivity and nonlinear response. Existing systems approaches to safety such as STPA offer some improvements, but are designed to deal with actively controlled systems, whereas dam systems have both active and passive controls. STPA is also unable to analyze reservoir level fluctuations in response to constraints. Stochastic simulation is the only technique that is able to determine reservoir level response to various constraints. It is also perhaps the easiest approach to estimate the overall system probability of overtopping failure. If run for enough years, a fully stochastic model, in theory, would eventually simulate the full range of potential operating state combinations. However, a fully stochastic simulation spends a significant amount of computational effort simulating non-failure states and would require a computationally prohibitive number of simulation-years to achieve a more thorough analysis of each possible combination of events. Ultimately, the ability of the traditional stochastic approach to analyze potentially threatening combinations of events is limited by the number of simulation-years – at the current time it is not possible to achieve a full coverage of all potential combinations of events using this method.

The methodology described in this research draws on the strengths of existing methods to more fully and systematically determine how the system will respond to as many combinations of events as can be determined. The operating states database and combinatorial procedure help to automate the process of determining potential constraints on system operation. The Deterministic Monte Carlo simulation framework systematically characterizes the potential system responses which may be expected for a given scenario. Scenarios are deterministic inputs to a simulation model that is run for a large number of iterations with Monte Carlo varying scenario parameters. The system dynamics simulation model is capable of representing as much complexity as desired in systems with component interaction and feedback in a transparent and easily modifiable object-oriented programming environment. The level of detail with which each aspect of the system is modelled also facilitates linking the model with the components database to enable the deterministic simulation of all of the possible combinations of operating states arrived at in the scenario generation procedure. Through system dynamics simulation, emergent

behaviour may be observed as a result of component interactions, with outcomes that may not be easily foreseeable by analysing sub-systems or parts of the system. Simulations automatically generate metrics such as the conditional failure frequencies and reservoir level exceedance frequencies for a particular scenario. While this research focuses on safety-specific indicators, it may also be possible to investigate environmental, regulatory and economic considerations from the simulation outcomes.

There are some limitations of the approach proposed in this thesis. Despite the much larger number of scenarios that this approach is able to generate and assess, the STPA analysis identified some scenarios that were not captured by the simulation. These scenarios involved operational decision making and process model errors, which would add another (very large) dimension to the simulation analysis – though it is theoretically possible to analyze such issues using simulation. The results from the case study illustrated how the number of possible scenarios increases exponentially with the number of components being modelled and the number of operating state-causal factor combinations. Ultimately, the methodology presented allows modellers build the system to as high a level of detail as is desired so that the key interactions and feedbacks are fully modelled. In applying the approach, however, this may result in computational feasibility challenges. It is possible that simplifications to improve computational efficiency could affect the outcomes of the analysis. Future work must address the computational requirements of fully-detailed models to ensure this approach can be extended to real dam systems. It is also not clear whether the consideration of each operating state-causal factor as a separate operating state is necessary – this introduces a fair amount of redundant simulation but was introduced to ensure causal factors and common cause failures were represented within the model. Finally, while the post-processing of scenarios helps to filter out scenarios that were not representative of the input scenario (ie. all events did not occur prior to dam failure, or all events did not affect one another), it may result in fewer data points than reasonable to estimate the conditional failure frequency and other criticality parameters.

The following section details some potential directions for future research.

5.2 Directions for Future Research

For future applications, it may be interesting to simulate the set of scenarios with starting reservoir elevations that would differ from “normal” conditions, in an attempt to model the system response to situations that could arise as a result of operational trade-offs that are difficult to generate automatically.

The most important area for future research relating to this approach is to incorporate probabilistic assessment into the approach. The methodology presented is set up fairly well to achieve this goal, in that the operating states database could be extended to allow estimates of the probabilities of causal factors and/or probabilities of component failure conditional on the causal factor occurring. These probabilities along with the Deterministic Monte-Carlo simulation results (conditional probabilities of failure or reservoir level exceedance) could be used in estimating the overall failure rate for each scenario. The benefit of the Deterministic Monte Carlo approach for assessment of overall overtopping failure probability is that the sensitivity to assumed operating state probabilities can be analyzed without significant additional computational effort. The full suite of results could be used to assess the probability of overtopping failure of the dam using traditional probability theory. The results would be a probability assessment that takes into account a far wider coverage of the possible operating states for the system than may be achieved using traditional techniques. The probability of flow control failure of the dam is an important decision-making factor for dam owners in terms of fleet management. Obviously, resources should be directed towards dams which have a higher probability of failure and/or a higher consequence category. In addition to this, the change in frequency of overtopping failure as a result of by different alternative operating strategies and capital upgrades could help provide a business case for investing in different alternatives (along with the visual aid of the aggregated scenario performance measure plots).

Another very important area for future work is in improving the computational feasibility of the approach described in this research. When the combinatorial procedure was applied to a detailed model of the Cheakamus System, 1.89×10^{27} scenarios resulted. This would obviously be computationally infeasible in a reasonable amount of time, though advances in computing capabilities may make it more feasible in the future. In the meantime, work

on applying the methodology to real systems with grouped components similar to what was done in the simplified system will help reduce the number of scenarios to a more realistic and computationally reasonable number. Grouping components could potentially be guided by a fault tree analysis of sub-systems within the system – for example, a fault tree analysis of the gate system to determine groups of components that might lead to a specific operating state for the gate as a whole. Implementing nested time-steps to address issues such as cavitation, pressure transients, erosion, slope stability and internal erosion would add additional complexity but is also an important area for future work. Further improvements to the simulation model speed may also be possible, although they would require a substantial effort and possibly a switch to a C++ or similar compiled programming language. Compiled programming languages are generally considered to be the most computationally efficient, however they are slightly less user friendly and require more programming experience.

Another potential direction for future work is the integration of the system dynamics simulation with AI to drive (or even build) the simulation model. Deep learning algorithms could potentially be applied to process the results to provide additional useful information from simulation outcomes. Finally, pattern recognition techniques may be useful to reduce the number of combinations required to assess each simulation outcome. This is a particularly promising direction that could help improve the limitation resulting from the trade-off between computational feasibility and level of complexity modelled.

Ultimately, these promising directions for future work may help to strengthen the approach, making it more readily applicable to existing, highly complex dam systems.

References

Adamo N, Al-Ansari N, Laue J, et al (2017) Risk Management Concepts in Dam Safety Evaluation: Mosul Dam as a Case Study. J Civ Eng Archit 11:635–652. https://doi.org/10.17265/1934-7359/2017.07.002

ASDSO (2018) Application of PFMA in dam safety. In: Webinar. https://learningcenter.damsafety.org/products/application-of-pfma-in-dam-safety-download. Accessed 24 Nov 2018.

Association of State Dam Safety Officials (ASDSO) (2010) Dam Failures, Dam Incidents (Near Failures). In: ASDSO Incident Database. https://damsafety.org/dam-failures. Accessed 12 Jul 2015.

Åström KJ, Murray RM (2008) Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press, Princeton, New Jersey

Baecher G, Ascila R, Hartford DND (2013) Hydropower and dam safety. In: STAMP/STPA Workshop. Cambridge, Massachusetts

Barker M, Vivian B, Bowles DS (2006) Reliability assessment for a spillway gate upgrade design in Queensland, Australia. In: USSD 2006 Conference. San Antonio, Texas

Barker M, Vivian B, Matthews J, Oliver P (2003) Spillway Gate Reliability and Handling of Risk for Radial and Drum Gates and Gate Operating. In: NZCOLD/ANCOLD 2003 Conference on Dams. pp 1–15

Bartsch M (2004) FMECA of the Ajaure Dam - A Methodology Study. In: 13th Biennial Conference, Long Term Benefits and Performance of Dams. Thomas Telford, Canterbury

BC Hydro (2005) Cheakamus Project Water Use Plan

BC Hydro (2013) Generation Operating Order: Cheakamus Project. Burnaby, BC

Bocchiola D, De Michele C, Rosso R (2003) Review of recent advances in index flood estimation. Hydrol Earth Syst Sci 7:283–296. https://doi.org/10.5194/hess-7-283-2003

Bowles D, Ruthford M, Anderson L (2011a) Risk assessment of success dam, California: Evaluation of operating restrictions as an interim measure to mitigate earthquake risk. In: GeoRisk 2011: Geotechnical Risk Assessment and Management. Atlanta, Georgia

Bowles DS (2001) Evaluation and Use of Risk Estimates in Dam Safety Decisionmaking. In: Risk-Based Decisionmaking in Water Resources IX. pp 17–32

Bowles DS, Anderson LR, Chauhan SS, et al (2015) Risk assessment approach for coal ash impoundments. In: ASDSO Conference on Dams. Louisville, Kentucky

Bowles DS, Anderson LR, Evelyn JB, et al (1999) Alamo dam demonstration risk assessment. In: ANCOLD Annual Meeting. Jindabyne, Australia, pp 113–128

Bowles DS, Anderson LR, Glover TF (1987) Design level risk assessment for dams. In: ASCE Seismic considerations in risk analysis of dams. pp 210–225

Bowles DS, Anderson LR, Glover TF (1998a) The practice of dam safety risk assessment and management: Its roots, its branches and its fruit. In: USCOLD Annual Meeting. Buffalo, New York, p 13

Bowles DS, Anderson LR, Glover TF, Chauhan SS (1998b) Portfolio Risk Assessment: a Tool for Dam Safety Risk Management. In: 1998 USCOLD Annual Lecture. Buffalo, New York, p 13

Bowles DS, Anderson LR, Ruthford ME, et al (2010) a Risk-Based Re-Evaluation of Reservoir Operating Restrictions To Reduce the Risk of Failure From Earthquake and Piping. In: ANCOLD Proceedings of Technical Groups. Hobart, Australia

Bowles DS, Chauhan SS, Anderson LR, Grove RC (2011b) Baseline risk assessment for herbert hoover dike. In: Association of State Dam Safety Officials Annual

Conference 2011, Dam Safety 2011

Breach P (2015) Python tools for Vensim simulation software. In: GitHub. https://github.com/pbreach/venpy. Accessed 3 Feb 2016

Brown GS, Campbell DP (1950) Instrument engineering: its growth and promise in process-control problems. Mech Eng 72:124–127

CDA (2007) Dam Safety Guidelines 2007. In: Dam Safety Publications. https://www.cda.ca/EN/Publications_Pages/Dam_Safety_Publications.aspx. Accessed 04 May 2015.

Chanson H (2000) A review of accidents and failures of stepped spillways and weirs. In: Proceedings of the Institution of Civil Engineers Water and Maritime Engineering. pp 177–188

Charles JA, Tedd P, Warren A (2011) Lessons from historical dam incidents. In: Flood and Coastal Erosion Risk Management Research and Development programme. Department for Environment Food and Rural Affairs. Bristol, UK.

Ching J, Leu S Sen (2009) Bayesian updating of reliability of civil infrastructure facilities based on condition-state data and fault-tree model. Reliab Eng Syst Saf 94:1962–1974. https://doi.org/10.1016/j.ress.2009.07.002

Cyganiewicz JM, Smart JD (2000) U. S. Bureau of Reclamation's use of risk analysis and risk assessment in dam safety decision making. In: ICOLD 20th Congress. Beijing, China, pp 1–19

Donnelly CR (2005) Assessing the safety and security of dams. In: International Conference on Safety and Security of Energy Infrastructures in a Comparative View. Brussels, Belgium, pp 1–34

dos Santos RNC, Caldeira LMMS, Serra JPB (2012) FMEA of a tailings dam. Georisk Assess Manag Risk Engneered Syst Geohazards 6:89–104. https://doi.org/10.1080/17499518.2011.615751

Duckworth HA, Moore RA (2010) Social Responsibility: Failure Modes Effects and Analysis. In: Industrial Innovation Series. Taylor and Francis, Abingdon, U. K.

Duscha LA, Jansen RB (1988) Surveillance. In: Jansen RB (ed) Advanced Dam Engineering for Design, Construction, and Rehabilitation. Van Nostrand Reinhold, New York, pp 777–799

Dusil R, To P (2016) Generation Water Passage – A System Perspective. In: Hydro Power Engineering Exchange. Queenstown, New Zealand, pp 1–16

El-Awady A (2019) Probabilistic Failure Analysis of Complex Systems with Case Studies in Nuclear and Hydropower Industries. PhD Thesis. University of Waterloo

Ellingwood BR (2017) Performance-based structural engineering in an era of climate change: A risk-informed approach. In: Attar A, Lounis Z (eds) NRCC Workshop on Climate Change and Code Adaptation. National Research Council of Canada, Ottawa, Ontario

Ericson CA (1999) Fault Tree Analysis – A History. In: Proceedings of The 17th International System Safety Conference. Orlando, Florida

Eum H-I, Simonovic SP (2012) Assessment on variability of extreme climate events for the Upper Thames basin in Canada. Hydrol Process 26:485–499

Ezell BC, Farr J V., Wiese I (2000) Infrastructure Risk Analysis Model. J Infrastruct Syst 6:114–117

Faber MH, Stewart MG (2003) Risk assessment for civil engineering facilities: Critical overview and discussion. Reliab Eng Syst Saf 80:173–184. https://doi.org/10.1016/S0951-8320(03)00027-9

FEMA (2004) Federal Guidelines for Dam Safety- Selecting and Accomodating Inflow Design Floods for Dams. Washington, D.C.

FEMA, NJOEM (2004) Findings of the Interagency Waterway Infrastructure

Improvement Task Force Report. Trenton, New Jersey.

FERC (2006) Report of Findings on the Overtopping and Embankment Breach of the Upper Dam - Taum Sauk Pumped Storage Project. Washington, D.C.

FERC (2005a) Oroville Dam: Potential failure mode analysis study report. Washington, D.C.

FERC (2005b) Dam Safety Performance Monitoring Program. Washington, D.C.

FERC (2007) Norway and Oakdale Hydroelectric Project: Baseline Risk Assessment Report. Washington, D.C.

Ferdous R, Khan F, Sadiq R, et al (2011) Fault and Event Tree Analyses for Process Systems Risk Analysis: Uncertainty Handling Formulations. Risk Anal 31:86–107. https://doi.org/10.1111/j.1539-6924.2010.01475.x

Forrester J (1969) Urban Dynamics. MIT Press, Cambridge, Massachusetts

Forrester JW (1961) Industrial Dynamics. Productivity Press, Portland, Oregon

Forrester JW (1971a) Principles of Systems. MIT Press, Cambridge, Massachusetts

Forrester JW (1971b) World Dyamics. Wright-Allan Press. Cambridge, Massachusetts

Forrester JW (1989) The beginings of System Dynamics. In: international meeting of the System Dynamics Society. Stuttgart, Germany, pp 1–16

Foster M, Fell R (2001) Assessing embankment dam filters that do not satisfy design criteria. J Geotech Geoenvironmental Eng 127:398–407

Foster M, Fell R, Spannagle M (2000a) The statistics of embankment dam failures and accidents. Can Geotech J 37:1000–1024. https://doi.org/10.1139/t00-030

Foster M, Fell R, Spannagle M (2000b) A method for assessing the relative likelihood of failure of embankment dams by piping: Reply. Can Geotech J 37:1025–1061. https://doi.org/10.1139/t01-109

France JW, Alvi IA, Dickson PA, et al (2018a) Independent Forensic Team Report Oroville Dam Spillway Incident

France M, Mutler J, Safar H (2018b) New Guidance for CAST: Case Study of a US Freight Rail Stop Signal Overrun & Collision. In: 2018 STAMP Workshop. Boston, Massachusetts

Fry J, Vogel A, Courivaud J-R, Blais J-P (2004) Dam Accident Data Base DADB - The Web Based Data Collection of ICOLD. In: Hewlett H (ed) 13th Conference of the British Dam Society and the ICOLD European Club. Thomas Telford Limited, Canterbury, UK, pp 298–304

Genevois R, Ghirotti M (2005) The 1963 Vaiont Landslide. G di Geol Appl 1:41–52. https://doi.org/10.1474/GGA.2005-01.0-05.0005

Goodarzi E (2010) Estimating Probability of Failure Due to Internal Erosion with Event Tree Analysis. Electron J Geotech Eng 15:935–948

Hansen KM, Ravn AP, Stavridou V (1998) From safety analysis to software requirements. IEEE Trans Softw Eng 24:

Hartford DND (2001) Risk Analysis In Geotechnical And Earthquake Engineering : State-Of-The-Art And Practice For Embankment Dams. In: Fourth International Conference on Recent Advances in Geotechnical Earthquake Engineering. San Diego, California

Hartford DND, Baecher GB (2004) Risk and Uncertainty in Dam Safety. Thomas Telford Limited, London

Hartford DND, Baecher GB, Zielinski PA, et al (2016) Operational Safety of Dams and Reservoirs. ICE Publishing, London

Hill P, Bowles D (2003) Estimating overall risk of dam failure: practical considerations in combining failure probabilities. ANCOLD 2003 Risk Work 10

Hill PI, Bowles DS, Nathan RJ, Herweynen R (2001) On the Art of Event Tree Modelling for Portfolio Risk Analysis. In: NZSOLD/ANCOLD Conference on Dams. pp 1–10

Hoeg K, Fry J-J, Charlwood R (2007) Peer Review of Dam Safety at Suorva Dams. Stockholm, Sweden

Hydrometrics Inc. (2011) Guidelines for Conducting a Simplified Failure Mode Analysis for Montana Dams. Helena, Montana

IEC (2008) Analysis techniques for system reliability - Procedure for failure mode and effects analysis (FMEA)

IEC (2010) Analysis techniques for dependability - Event tree analysis (ETA)

Jansen RB (1983) Dams and Public Safety. United States Department of the Interior, Denver, Colorado

Jong CG, Leu S Sen (2013) Bayesian-network-based hydro-power fault diagnosis system development by fault tree transformation. J Mar Sci Technol 21:367–379. https://doi.org/10.6119/JMST-012-0508-3

Kast FE, Rosenzweig JE (1972) General systems theory: Applications for organization and management. Acad Manag J 15:447–465. https://doi.org/10.5465/255141

Kennard MF, Bromhead EN (2000) Carsington Dam - The near-miss which became a bulls-eye. Forensic Eng 102–111

King LM, Keech S, Simonovic SP (2016a) An Investigation of the Factors and Components Involved in Dam Safety Flow Control Incidents. J Dam Eng 27:1–19

King LM, Mcleod AI, Simonovic SP (2015) Improved Weather Generator Algorithm for Multisite Simulation of Precipitation and Temperature. J Am Water Resour Assoc 51:. https://doi.org/10.1111/1752-1688.12307

King LM, Mcleod AI, Simonovic SP (2014) Simulation of historical temperatures using a

multi-site, multivariate block resampling algorithm with perturbation. Hydrol Process 28:. https://doi.org/10.1002/hyp.9596

King LM, Simonovic SP (2020) A Deterministic Monte Carlo Simulation Framework for Dam Safety Flow Control Assessment. Water 12(2):505. https://doi.org/10.3390/w12020505

King LM, Simonovic SP, Hartford DND (2017) Using system dynamics simulation for assessment of hydropower system safety. Water Resour Res 53:. https://doi.org/10.1002/2017WR020834

King LM, Simonovic SP, Hartford DND (2016b) A hydropower infrastructure simulation model for assessment of resilience. In: Canadian Society of Civil Engineering 2016 National Conference. London, pp 1–11

Komey A (2014) A systems reliability approach to flow control in dam safety risk analysis. MESc Thesis. University of Maryland College Park

Komey A, Deng Q, Baecher GB, et al (2015) Systems Reliability of Flow Control in Dam Safety. In: 12th International Conference on Application of Statistics and Probability in Civil Engineering, ICASP12. Vancouver, British Columbia, pp 1–8

Kotz S, van Dorp JR (2004) The Triangular Distribution. In: Beyond Beta: Other continuous families of distributions with bounded support and applications. World Scientific Publishing Co. Ptc. Ltd., Singapore, pp 1–32

Kuo J-T, Yen B-C, Hsu Y-C, Lin H-F (2007) Risk Analysis for Dam Overtopping—Feitsui Reservoir as a Case Study. J Hydraul Eng 133:955–963. https://doi.org/10.1061/(ASCE)0733-9429(2007)133:8(955)

Kwon HH, Moon YI (2006) Improvement of overtopping risk evaluations using probabilistic concepts for existing dams. Stoch Environ Res Risk Assess 20:223–237. https://doi.org/10.1007/s00477-005-0017-2

Lee WS, Grosh DL, Tillman FA, Lie CH (1985) Fault Tree Analysis, Methods, and

Applications - A Review. IEEE Trans Reliab R-34:194–203. https://doi.org/10.1109/TR.1985.5222114

Leveson N, Daouk M, Dulac N, Marais K (2003) Applying STAMP in accident analysis. In: Workshop on Investigating and Reporting of Incidents and Accidents. Williamsburg, Virginia, pp 177–198

Leveson NG (1995) Safeware: System Safety and Computers. Addison-Wesley, Boston, Massachusetts

Leveson NG (2011) Engineering a Safer World: Systems Thinking Applied to Safety. The MIT Press, Cambridge, Massachusetts

Lewin J, Ballard G, Bowles DS (2003) Spillway Gate Reliability in the Context of Overall Dam Failure Risk. In: 2003 USSD Annual Lecture. Charleston, South Carolina, pp 1–17

Lin C-T, Wang M-J (1997) Hybrid fault tree analysis using fuzzy sets. Reliab Eng Syst Saf 58:205-213

Mandal S, Breach PA, Guar A, Simonovic SP (2017) Tools for downscaling climate variables: A technical manual. London, U.K.

McDonald L, Wan CF (1999) Risk assessment for Hume Dam - Lessons from estimating the chance of failure. In: ANCOLD Conference on Dams. Jindabyne, Australia, pp 11–24

Mcgrath S (2000) To Study International Practice and Use of Risk Assessment in Dam Management. In: The Winston Churchill memorial trust of Australia. https://d1rkab7tlqy5f1.cloudfront.net/TBM/Over%20faculteit/Afdelingen/Values%2C%20Technology%20and%20Innovation/People/Full%20Professors/Pieter%20van%20Gelder/Citations/citatie20.pdf. Accessed 12 Sept 2019.

Micovic Z, Hartford DND, Schaefer MG, Barker BL (2015) Flood hazard for dam safety - Where the focus should be? In: 25th ICOLD Congress. ICOLD, Stavanger,

Norway, p 22

Micovic Z, Quick MC (1999) A rainfall and snowmelt runoff modelling approach to flow estimation at ungauged sites in British Columbia. J Hydrol 226:101–120

Morris M, Wallis M, Brown A, et al (2012) Reservoir safety risk assessment – a new guide. In: British Dam Society Annual Conference. Leeds, U.K., pp 1–10

National Research Council (1981) Safety and offshore oil. National Academy Press, Washington, D.C.

NPDP (2016) National Performance of Dams Program, Stanford University. http://npdp.stanford.edu/. Accessed 7 Jan 2016

Patev RC, Putcha C, Foltz SD (2005) Methodology for Risk Analysis of Dam Gates and Associated Operating Equipment Using Fault Tree Analysis. Washington, D.C.

Pope G, Breneman J (2018) STPA for use by compiler and binary analysis tools. In: 2018 STAMP Workshop. Boston, Massachusetts

Prodanovic P, Simonovic SP (2008) Intensity duration frequency analysis under changing climatic conditions. In: 4th International Symposium on Flood Defense: Managing Flood Risk, Reliability and Vulnerability. Toronto, Ontario, pp 1–8

Putcha CS, Patev RC (2000) Investigation of Risk Assessment Methodology for Dam Gates and Associated Operating Equipment. USACE, Washington, D.C.

Python Software Foundation (2012) Functions creating iterators for efficient looping. In: Python Stand. Libr. https://docs.python.org/3/library/itertools.html. Accessed 14 May 2018.

Quick MC, Pipes A (1977) U.B.C Watershed Model. Hydrol Sci J 22:153–161. https://doi.org/10.1080/02626667709491701

Rausand M, Hoyland A (2004) System Reliability Theory: Models, Statistical Methods, and Applications, 2nd edn. John Wiley and Sons, Hoboken, New Jersey

Regan PJ (2009) Dam failures vs. age of dams. In: 29th Annual USSD Conference. Nashville, Tennessee

Regan PJ (2010) Dams as systems - A holistic approach to dam safety. In: USSD Annual Meeting and Conference. Sacramento, California, pp 1307–1340

Richardson GP (1991) Feedback Thought in Social Science and Systems Theory. University of Pennsylvania Press, Philadelphia

Rychkov V, Kawahara K (2015) ADAPT-MAAP4 coupling for a dynamic event tree study. In: International Topical Meeting on Probabilistic Safety Assessment and Analysis. Sun Valley, Idaho, pp 140–143

SAE (1996) Guidelines and methods for conducting the safety assessment process on civil airborne systems and equipment.

SAE (1967) Design analysis procedure for Failure Modes, Effects and Criticality Analysis

Schmittner C, Gruber T, Puschner P, Schoitsch E (2014) Security Application of Failure Mode and Effect Analysis (FMEA). In: International Conference on Computer Safety, Reliability, and Security. Firenze, Italy, pp 310–325

Seed HB, Duncan JM (1987) The faiure of Teton Dam. Eng Geol 24:173–205

Sharif M, Burn DH (2006) Simulating climate change scenarios using an improved K-nearest neighbor model. J Hydrol 325:179–196. https://doi.org/10.1016/j.jhydrol.2005.10.015

Shaw G, Fan BH, Hartford DND (2000) Seismic Safety Assessment of Two Concrete Dams in a Cascade Development-Investigations Into the Use of Qualitative Risk. In: 12th World Conference on Earthquake Engineering. Auckland, New Zealand, pp 1–8

Simonovic SP (2009) Managing Water Resources. UNESCO, London

Skelton B (1997) Process safety analysis: An introduction. Gulf Pub, Houston, Texas

Smith M (2006) Dam Risk Analysis Using Bayesian Networks. In: 2006 ECI Conference on Geohazards. Lillehammer, Norway

Song Y (2012) Applying System-Theoretic Accident Model and Processes (STAMP) to Hazard Analysis. MESc Thesis. McMaster University

SPANCOLD (2012) Risk analysis applied to management of dam safety. In: Technical Guides on Dam Safety. Madrid, Spain.

Srivastava A (2008) Generalized Event Tree Algorithm and Software for Dam Safety Risk Assessment. MESc Thesis. Utah State University

Srivastava A (2013) A Computational Framework for Dam Safety Risk Assessment with Uncertainty Analysis. PhD Thesis. Utah State University

Srivastava A, Bowles DS, Chauhan SS (2012) Damrae-U: A tool for including uncertainty in dam safety risk assessment. In: Association of State Dam Safety Officials Annual Conference 2012. Denver, Colorado

Stamatis DH (2002) Design for six sigma. In: Six sigma and beyond. St Luice Press, Boca Raton, Florida

Stedinger J, Heath D, Nagarwalla N (1989) Event tree simulation analysis for dam safety problems risk analysis. In: Risk Analysis and Management of Natural and Man-Made Hazards. Santa Barbara, California

Sterman JD (2000) Business Dynamics: Systems thinking and modeling for a complex world. McGraw Hill, Boston, Massachusetts

Stewart RA (2000) Dam risk management. In: Proceedings of the International Conference on Geotechnical & Geological Engineering (GeoEng 2000). Melbourne, Australia, pp 19–24

Tavakoli N (2015) Comprehensive literature review on dam overtopping incidents. In:

Toledo MÁ, Morán R, Oñate E (eds) Dam Protections against Overtopping and Accidental Leakage. Taylor and Francis Group, London, pp 169–180

Thomas J (2012) Extending and Automating a Systems- Theoretic Hazard Analysis for Requirements Generation and Analysis. Sandia National Laboratories. Albuquerque, New Mexico.

Thomas J (2013) Extending and automating a Systems-Theoretic hazard analysis for requirements generation and analysis. PhD Thesis. Massachusetts Institute of Technology

To P, Dusil R, Hydro BC (2018) STPA Application in Hydropower – Piloting Experience. In: 2018 STAMP Workshop. Boston, Massachusetts

Todd R V (1999) Spillway tainter gate failure at Folsom Dam, California. In: Waterpower Conference 1999. Las Vegas, pp 1–10

Tummala R, Schoenherr T (2011) Assessing and managing risks using the Supply Chain Risk Management Process (SCRMP). Supply Chain Manag 16:474–483. https://doi.org/10.1108/13598541111171165

USACE (2011) Safety of Dams - Policy and Procedures. Washington, D.C.

USBR (1987) Design of Small Dams, Third Edition. United States Department of the Interior, Washington, D.C.

USBR (2014a) RCEM Reclamation Consequence Estimating Methodology: Dam failure and flood event case history compilation. Denver, Colorado.

USBR (2014b) Design Standards No. 14: Appurtenant Structures for Dams (Spillways and Outlet Works). Washington, D.C.

USBR (2011) Dam Safety Public Protection Guidelines. Denver, Colorado

USBR (2018) Hydrologic Hazard Analysis. Denver, Colorado.

USBR, USACE (2012) Hydrologic Hazard Analysis. In: Dam Safety Risk Analysis Best Practices Training Manual. United States Department of the Interior, Denver Colorado, pp 7.1-7.12

USBR, USACE (2015a) Best Practices Training Manual. United States Department of the Interior, Denver Colorado

USBR, USACE (2015b) Potential Failure Mode Analysis. Washington, D.C.

USSD (2013) Routine Instrumented and Visual Monitoring of Dams Based on Potential Failure Modes Analysis. Denver, Colorado

Van Niekerk HJ, Viljoen MJ (2005) Causes and consequences of the Merriespruit and other tailings-dam failures. L Degrad Dev 16:201–212. https://doi.org/10.1002/ldr.681

Ventana Systems (2015) Vensim. https://vensim.com/. Accessed 5 Jan 2017

Vernacchia MA (2018) Use of STPA Within the GM System Safety Process. In: 2018 STAMP Workshop. Boston, Massachusetts

Vick SG (1992) Risk in geotechnical practice. In: Geotechnique and natural hazards. BiTech Publishers, Vancouver, British Columbia

Villa V, Cozzani V (2016) Application of Bayesian Networks to quantitative assessment of safety barriers' performance in the prevention of major accidents. Chem Eng Trans 53:151–156. https://doi.org/10.3303/CET1653026

von Bertalanffy L (1968) General System Theory: Foundations, Development, Applications. George Brazillier, Inc., New York

Von Bertalanffy L (1950) The theory of open systems in physics and biology. Science (80-) 111:23–29. https://doi.org/10.1126/science.111.2872.23

Whitman R V (1984) Evaluating calculated risk in geotechnical engineering. J Geotech Engrg 110:143–188

Wieland M, Malla S, Speerli J, et al (2005) Risk analysis of dam and powerhouse structure of run-of-river powerplant in Latvia. In: 73rd Annual Meeting of ICOLD. Tehran, Iran, p 10

Wiener N (1948) Cybernetics: or Control and Communication in the Animal and the Machine. MIT Press, Cambridge, Massachusetts

Yegian BMK, Marciano EA, Ghahraman VG (1991) Seismic risk analysis for earth dams. J Geotech Eng 117:18–34

Zhang L, Peng M, Chang D, Xu Y (2018) Analysis of Probability of Failure of Dams. In: Dam Failure Mechanisms and Risk Assessment, 2016 edn. Singapore, pp 243–272

Zhang LM, Xu Y, Jia JS (2007) Analysis of earth dam failures: A database approach. In: First International Symposium on Geotechnical Safety & Risk. Shanghai, China, pp 293–302

Zhang LM, Xu Y, Jia JS, Zhao C (2011) Diagnosis of embankment dam distresses using Bayesian networks. Part 1. Global-level characteristics based on dam distress database. Can Geotech J 48:1630–1644. https://doi.org/10.1139/T11-069

Zielinski PA, Perdikaris J, Zhou R, Sakamoto D (2016) Workshop E: Systems approach and simulation in risk assessment of dams. In: Canadian Dam Association Conference and Exhibition. Halifax, Nova Scotia

Appendix A: Cheakamus Hydropower Project Details

This appendix contains the numerical relationships for reservoir storage and flow conveyance at Cheakamus Dam. These relationships can be found in the publicly available Water Use Plan.

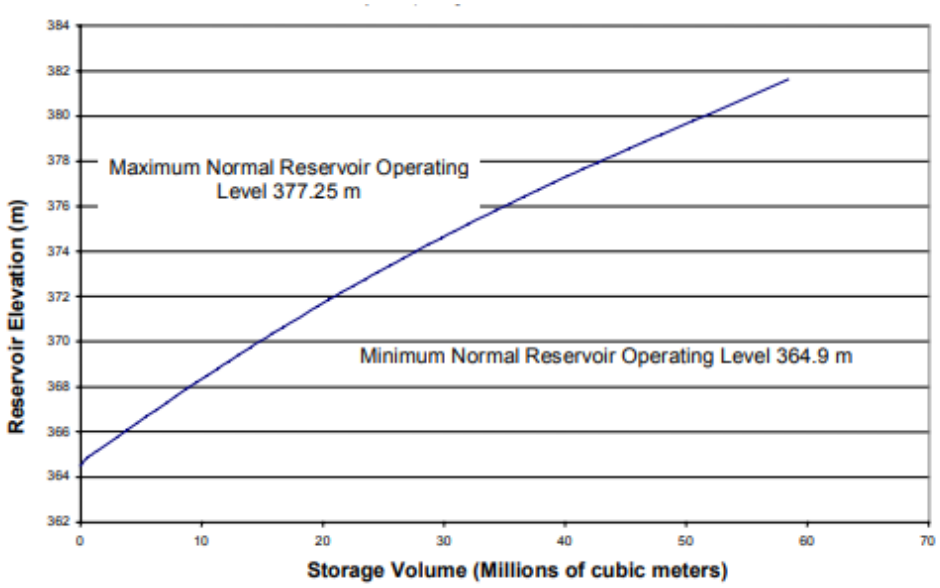


Figure A1 Stage-Storage curve for Daisy Lake Reservoir (BC Hydro 2005)

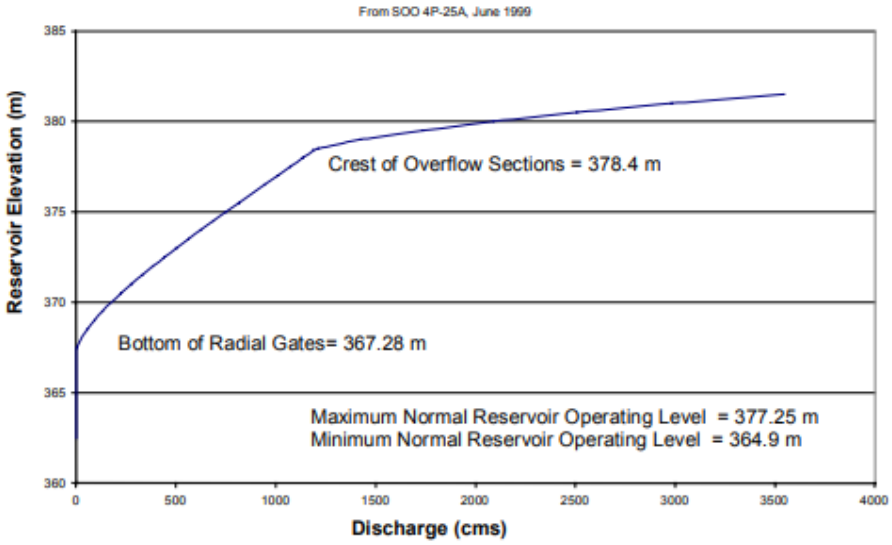


Figure A2: Rating Curve for radial gates fully open and free overflow (BC Hydro 2005)

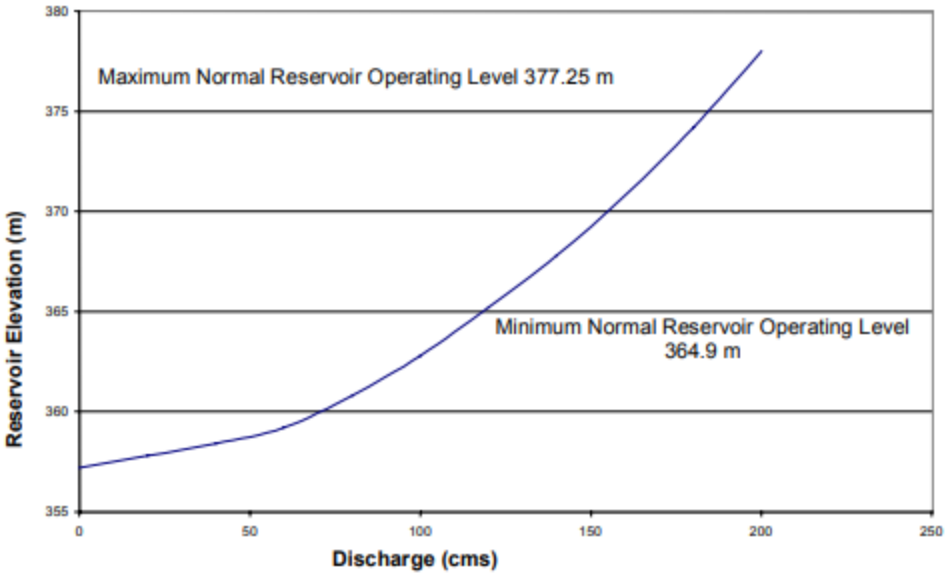


Figure A3: Rating curve for low level outlet sluice gate fully open (BC Hydro 2005)

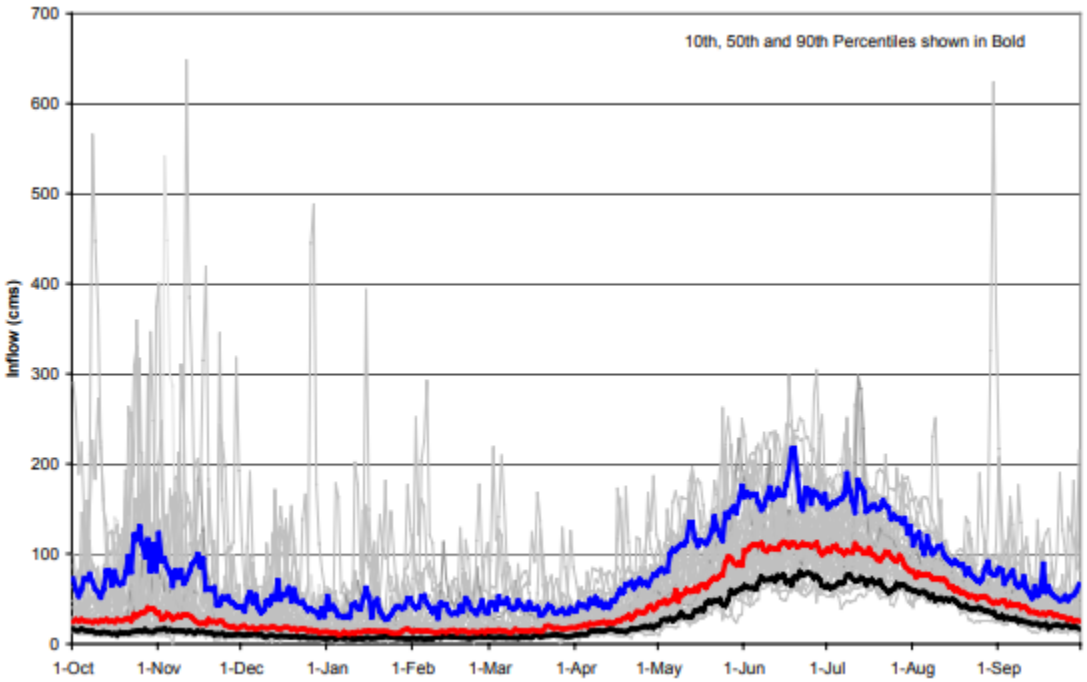


Figure A4: Historical Inflows to Daisy Lake (BC Hydro 2005)

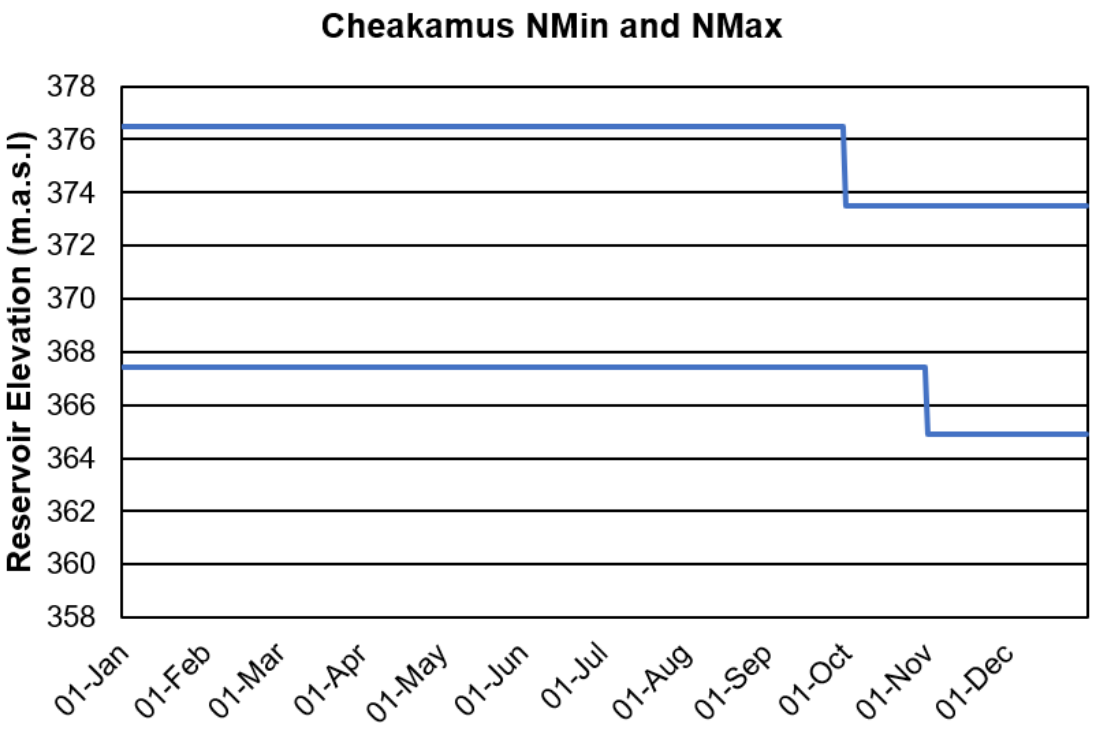


Figure A5: Daisy Lake NMin and NMax reservoir levels (data from BC Hydro 2005)

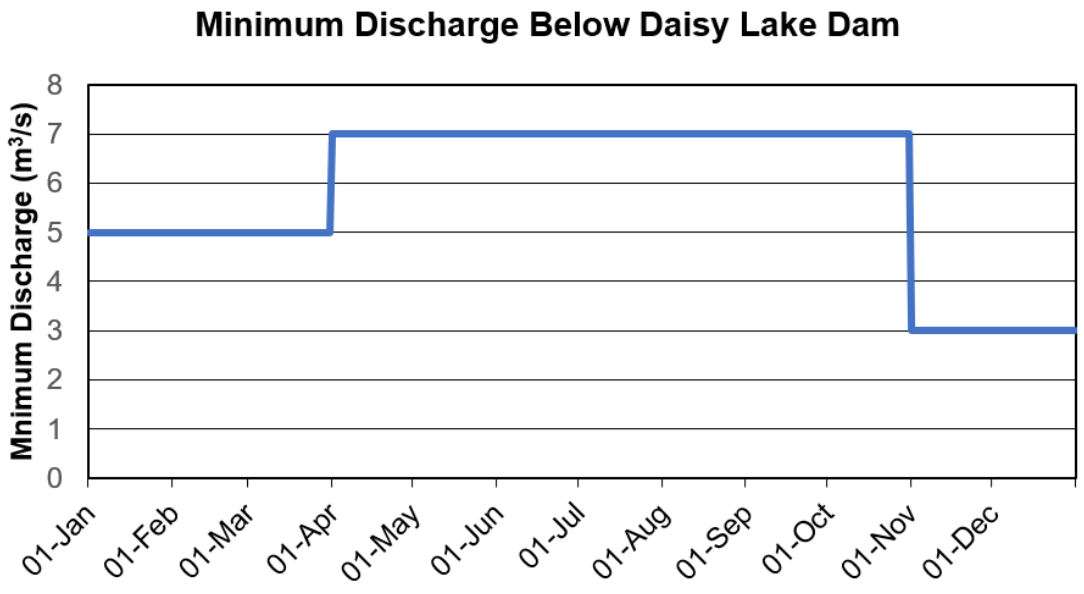


Figure A6: Minimum discharges below Daisy Lake Dam (data from BC Hydro 2005)

Appendix B: STPA Analysis of Cheakamus Dam

Cheakamus Dam System STPA analysis

Note that this analysis was not done by BC Hydro personnel and therefore should not be interpreted to represent real conclusions for the Cheakamus System.

High-level system hazards:

H1: High flows released into Cheakamus River and/or Squamish River (flood)

H2: Flow releases to Cheakamus River stopped (fish kill)

H3: Equipment damaged (economic/safety impact)

H4: Loss of power production (economic impact)

> -Not sure about this one. This happens quite often, during low flows or during maintenance, is it really to be considered an accident? I've decided to remove it from the list because it's a pretty regular occurrence. It might be more suitable to be added to the list for a large dam where water is always passing through the powerhouse (eg. Mica, Revelstoke, GMS)

High-level system safety constraints (requirements):

SH1: Flows released into Cheakamus and/or Squamish must not exceed a level that causes damage downstream (unless inflows do?)

SH2: Flow must always be released to Cheakamus River

SH3: Equipment must not become damaged

System Control Structure

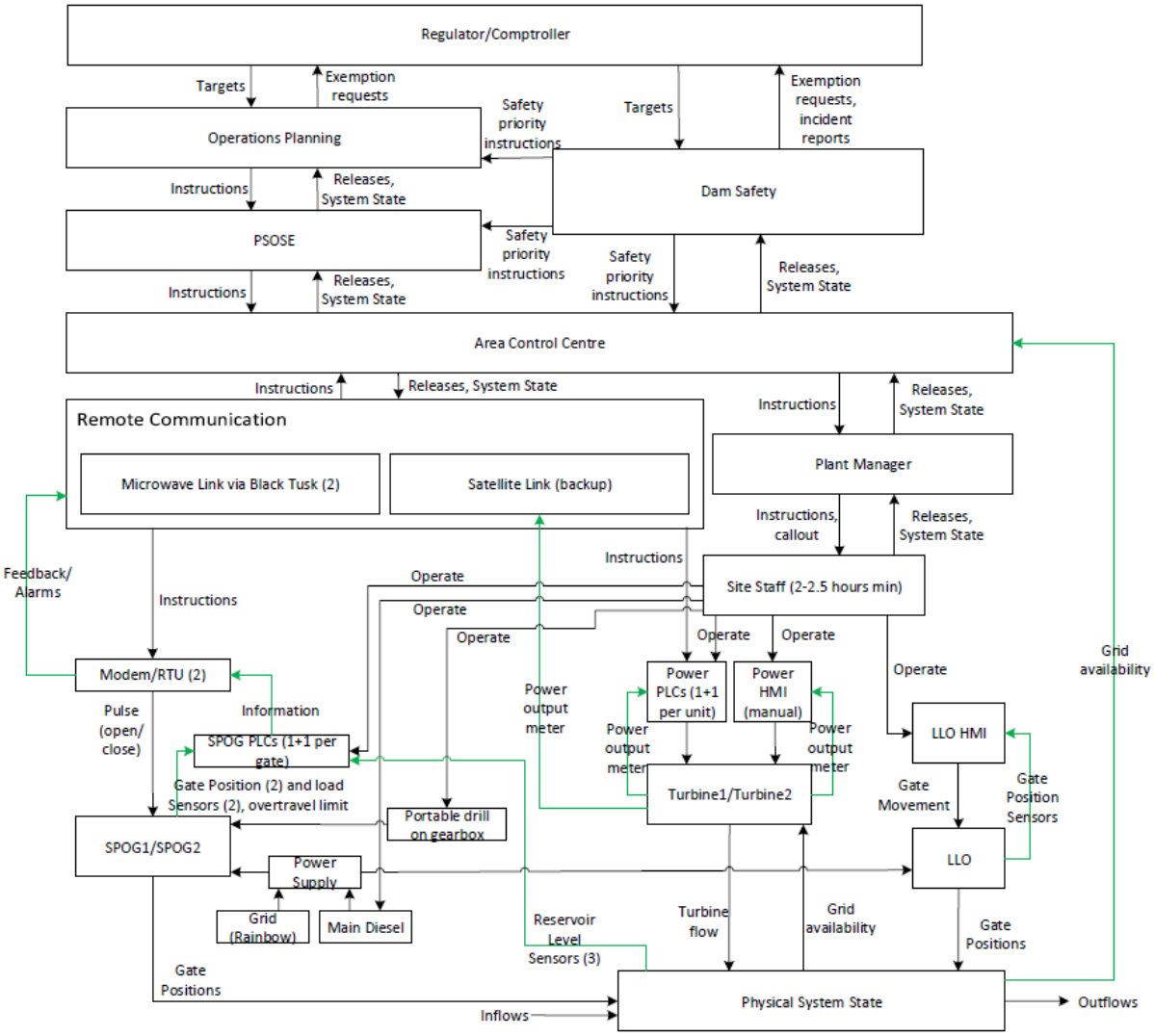


Figure B1: Hierarchical system structure

Table B1: STEP 1: Unsafe control actions

#	Control Action	Not Providing causes hazard	Providing causes hazard	Wrong timing/order causes hazard	Stopping too soon/applying too long causes hazard
1	SPOG1 Open	-UCA1 Open command not provided when water level high, inflow high or both [H1, H3] -UCA2 Open command not provided when SPOG2 and LLO closed [H2] -UCA3 Open command not provided but gate opens on its own [H1]	-UCA4 Open command provided resulting in downstream flooding [H1] -UCA5 Open command provided when gates blocked with debris/ice [H1, H3] -UCA6 Open command provided but gate stays closed [H2, H3]	-UCA7 Open command provided too late, after reservoir filled to unsafe level and/or gates overtopped [H1, H3]	-UCA8 Gate not left open long enough, reservoir continues to rise [H1, H3] -UCA9 Gate left open too long, resulting in draining of the reservoir to gate sill and fish kill [H2]
2	SPOG2 Open	-UCA10 Open command not	-UCA13 Open command	-UCA16 Open command	-UCA17 Gate not left open

		provided when water level high, inflow high or both [H1, H3] -UCA11 Open command not provided when SPOG1 and LLO closed [H2] -UCA12 Open command not provided but gate opens on its own [H1]	provided resulting in downstream flooding [H1] -UCA14 Open command provided when gates blocked with debris/ice [H1, H3] -UCA15 Open command provided but gate stays closed [H2, H3]	provided too late, after reservoir filled to unsafe level and/or gates overtopped [H1, H3]	long enough, reservoir continues to rise [H1, H3] -UCA18 Gate left open too long, resulting in draining of the reservoir to gate sill and fish kill [H2]
3	SPOG1 Close	-UCA19 Close command not provided when inflows and water level low (approaching sill) [H2] -UCA20 Close command not	-UCA21 Close command provided when reservoir and/or inflows high [H1, H3] -UCA22 Close command	-UCA23 Close command provided too early, reservoir level increases [H1, H3] -UCA24 Close command	

		provided when SPOG2 and/or LLO releasing excess water [H1] UCA25 Close command not provided, gate closes on its own [H1, H2, H3]	provided when SPOG2 and LLO closed [H2]	provided too late, reservoir drains [H2]	
4	SPOG2 Close	-UCA26 Close command not provided but gate closes on its own [H1, H2, H3] -UCA27 Close command not provided when inflows and water level low (approaching sill) [H2]	-UCA29 Close command provided when reservoir and/or inflows high [H1, H3] -UCA30 Close command provided when SPOG1 and LLO closed [H2]	-UCA31 Close command provided too early, reservoir level increases [H1, H3] -UCA32 Close command provided too late, reservoir drains [H2]	

		-UCA28 Close command not provided when SPOG1 and/or LLO releasing excess water [H1]			
5	LLO Open	-UCA33 Open command not provided when SPOGs closed [H1, H2, H3]	-UCA34 Open command provided when SPOGs open [H1]	-UCA35 Open command provided too late [H1, H2, H3]	UCA36 Open command provided too long, reservoir drains [H2] -UCA37 Open command stopped too early, reservoir rises [H1, H3]
6	LLO Close	-UCA38 Close command not provided when reservoir level approaching sill [H2]	UCA39: Close command provided when inflows and reservoir elevation high [H1, H2, H3]	UCA40: Close command provided too early [H1, H2, H3] UCA41: Close command	

				provided too late [H2]	
7	T1 Open	-UCA42 Open command not provided when inflows high and SPOGs out of service [H1, H3]	-UCA43 Open command provided but does not work, SPOGs out of service [H1, H3]	-UCA44 Open command provided too late, SPOGs out of service and inflow high [H1, H3]	-UCA45 Open command provided too long, reservoir level falls below power intake sill
8	T2 Open	-UCA46 Open command not provided when inflows high and SPOGs out of service [H1, H3]	-UCA47 Open command provided but does not work, SPOGs out of service [H1, H3]	-UCA48 Open command provided too late, SPOGs out of service and inflow high [H1, H3]	-UCA49 Open command provided too long, reservoir level falls below power intake sill
9	T1 Close	-UCA50 Close command not provided when reservoir levels low [H2]	-UCA52 Close command provided when reservoir levels high [H1, H3]	-UCA53 Close command provided too late, reservoir level low [H2]	

		-UCA51 Turbine closes when command not provided causing water hammer and penstock rupture [H3]			
10	T2 Close	-UCA54 Close command not provided when reservoir levels low [H2] -UCA55 Turbine closes when command not provided causing water hammer and penstock rupture [H2, H3]	-UCA56 Close command provided when reservoir levels high [H1, H3]	-UCA57 Close command provided too late, reservoir level low [H2]	

STEP 2: Causes of unsafe control actions

NOTE: additional details provided about recurring potential issues at end of this section.

UCA1/UCA10: SPOG Open command not provided when water level high, inflow high or both [H1, H3]

Case 1: Water level high, inflow low, open command not provided

-Controllers (OP, PSOSE, ACC, DS) unaware of reservoir level due to gauge failure, sensor failures or communication delays

-High tides at Squamish mean there are flooding impacts when additional flows are released from the CMS system. Controllers (OP, PSOSE, ACC, DC) make a decision to hold water back, allowing the reservoir to rise to an unsafe state even though the inflow is relatively low.

Case 2: Water level high, inflow high, open command not provided

-Controllers (OP, PSOSE, ACC) believe they can return the reservoir to a safe level using the powerhouse and/or LLO and/or other SPOG due to inflow forecast errors

-Controllers (OP, PSOSE, ACC, DS) unaware of reservoir level due to gauge failure, sensor failures or communication delays

-High tides at Squamish mean there are flooding impacts when additional flows are released from the CMS system. Controllers (OP, PSOSE, ACC, DC) make a decision to hold water back, allowing the reservoir to rise to unsafe levels

-Controllers do not follow procedure (human error due to fatigue or shift change at PSOSE/FVO)

Case 3: Water level low, inflow high, open command not provided

-Controllers (OP, PSOSE, ACC) believe they can keep the reservoir at a safe level without opening the gate, due to inflow forecast errors or process errors

-Gate(s) out of service for maintenance purposes and therefore cannot be opened.

-Controller thinks gate open (sensor failure, communication delay)

UCA2/UCA11: SPOG Open command not provided when other SPOG and LLO closed [H2]

-Procedural: Inflow low and operators want to conserve water for power production

-Controller (ACC) makes a mistake due to being tired or shift change

-Controller thinks gate open (sensor failure, communication delay)

UCA3/UCA12: SPOG Open command not provided but gate opens on its own [H1]

-Gate position sensor failure causes PLC to open gate spuriously

UCA4/UCA13: SPOG Open command provided resulting in downstream flooding [H1]

-Controllers (OP, PSOSE, ACC) issue a command to open the gate to a large opening, resulting in downstream flooding

-Inflow forecast error (controller thinks inflow is going to be higher than it is)

-Sensor failure or delay resulting in operator thinking reservoir level/inflow is higher than it actually is and opening gate to return it to a safe level

-Controllers issue a command to open the gate and the force is too high, alarm sent and ignored by controller, or sensor fails and alarm not sent - so the steel yields and there is an uncontrolled release of water. Note – hoist likely to fail first.

-Backup motor engages when main motor functional due to missing signal, two motors functioning to move gears causing overforce on hoist, hoist failure and gate fails closed. If inflows high enough, this could potentially result in dam breach.

UCA5/UCA14: SPOG Open command provided when gates blocked with debris or ice [H1, H2, H3]

-Operator (remote or manual) tries to open gate but it is blocked by debris/ice and a gate component fails (hoist, motor, strut) [H3]

-Gate fails open and a large amount of water is released [H1]

-Gate fails closed and no water is released [H2]

-Could potentially result in [H1] if inflows high enough

-Operator (remote or manual) opens gate but debris or ice result in less water being released than intended. Reservoir then rises to an unsafe level and excessive flows are discharged via free overflow spillway and/or over the dam crest [H1]

-Gate opens as planned and debris flushed through spillway chute damages chute [H3]

UCA6/UCA15: SPOG Open command provided but gate stays closed [H2, H3]

-Gate component has failed (deterioration or disturbance), resulting in inability to move gate

-Unable to move due to inadequate lubrication of guidewall/skinplate interface or trunnion

-Gate sensor failed and PLC thinks gate is open so it doesn't move the gate (or gate sensor doing it's job but debris in the way/some sort of interruption that makes the sensor sense the wrong position)

-Grid failed, diesel backup failed, site staff unavailable to operate temporary diesel generator

-Emergency situation such as an earthquake/landslide, or site inaccessibility

UCA7/UCA16: SPOG Open command provided too late, after reservoir filled to unsafe level and/or gates overtopped [H1, H3]

-Inflow forecast errors so controller doesn't think gate needs to be opened.

-Sensor errors so controller doesn't realize reservoir level is high

-Reservoir rises to above gates which can then no longer be opened and may be damaged [H1, H3]

-Reservoir rises to above gates which may still be opened but too late to prevent dangerous releases over free overflow and/or dam crests [H1, H3]

-Gate cannot be opened past 2m remotely. Delay in mobilizing site staff leads to unsafe conditions (access road issues/short staffed due to time of evening or weekend, inflows high and staff need ~2h minimum to reach site from Mission office)

-High tides at Squamish mean there are flooding impacts when additional flows are released from the CMS system. Controllers (OP, PSOSE, ACC, DC) make a decision to hold water back, allowing the reservoir to rise to an unsafe state even though the inflow is relatively low. Dam safety controller (DS) steps in at a certain reservoir elevation to override operations planners but by then it is too late

UCA8/UCA17: SPOG Gate not left open long enough, reservoir continues to rise [H1, H3]

-Controller thinks reservoir is lower (sensor failure) so they close gate

-Controller thinks inflows are manageable with other release facilities (inflow forecast/process model error) so they close the gate

-Gate closes on its own due to failure of some gate component (gate fails closed)

UCA9/UCA18: SPOG Gate left open too long, resulting in draining of the reservoir to gate sill and fish kill [H2]

-Controller thinks reservoir is higher (sensor failure/delay/relay failure) so they keep the gate open

-Loss of power to close gate (grid, diesel), temporary diesel requiring staff mobilization which has some delay

-Loss of site access and remote gate control meaning gate cannot be closed until site accessed or sat/microwave links working

-Gate fails in open position

-Loss of remote gate control, mobilization of staff to site to close gate takes too long and reservoir drains below sill

-UCA19/UCA27 SPOG Close command not provided when inflows and water level low (approaching sill) [H2]

-Reservoir level gauge faulty or delayed so controller thinks reservoir is higher than it is and drains it to the sill

-Controller thinks inflows are high (inflow forecast error) and keeps gate open despite plummeting reservoir elevation

-Operator lowering reservoir following signs of internal erosion or earthquake

-UCA20/UCA28 SPOG Close command not provided when other SPOG and/or LLO releasing excess water [H1]

-Process model: Controller thinks inflows high so keeps outflows high (past experience/ inflow forecast error)

-Process model: Controller thinks reservoir is high so keeps outflows high (sensor error, past experience with flashy reservoir)

-Inflows and reservoir level high, controller following procedure

-Lowering reservoir due to signs of internal erosion/damaged dam(s)

-UCA25/UCA26 SPOG Close command not provided, gate closes on its own [H1, H2, H3]

-Gate fails closed (failure of hoist, connections, structural, sensor, etc.)

-Spurious closure due to faulty gate position sensor

-UCA21/UCA29 SPOG Close command provided when reservoir and/or inflows high [H1, H3]

-Case 1: Reservoir high, inflows low

-Controller relies on past experience and thinks the situation can be handled with minimum fish flow and maximum power flow releases. If the reservoir is above the level of the earth dam filter, this decision could put the dam at risk of failure.

-Controller believes reservoir is low (gauge failure or delay, relay failure)

-Case 2: Reservoir low, inflows high

-Controller wants to fill reservoir higher to conserve water for energy production. Eventually if inflows stay high, this could mean larger spills later [H1] or even put the dam at risk in extreme cases [H1, H3]

-Controller believes inflows are low (inflow forecast error)

- High tides at Squamish and high tributary flows in Squamish and lower Cheakamus mean controller opts to hold water back to prevent flooding.

-Case 3: Reservoir high, inflows high

-High tides at Squamish and high tributary flows in Squamish and lower Cheakamus mean controller opts to hold water back to prevent flooding.

-Controller believes inflows are low (inflow forecast error) and believes power and minimum fish flow discharge will be sufficient to return water level to safe state

-Controller believes reservoir is low (gauge failure or delay, relay failure) and wants to fill to higher level to use the water for power production

-Spillway chute becomes damaged (debris? Or age) and operator wants to avoid further damage to chute so the gates are closed and the reservoir is allowed to rise to the level of the free overflow spillway. If inflows are high enough, this could potentially put the dam at risk of overtopping. Could also cause damage to and/or undermining of saddle/wing dams

-UCA22/UCA30 SPOG Close command provided when other SPOG and LLO closed [H2]

-Controller thinks water is being released through SPOG and/or LLO for fish flows (faulty SPOG position gauge and/or delay in information from on-site operator of LLO)

-ACC controller accidentally sends command to close gate

-UCA23/UCA31 SPOG Close command provided too early, reservoir level increases [H1, H3]

-Controller thinks reservoir level is low (gauge failure or delay)

-Controller thinks inflow is low (inflow forecast error)

-High tides at Squamish mean there are flooding impacts when additional flows are released from the CMS system. Controllers (OP, PSOSE, ACC, DC) make a decision to hold water back, closing the SPOGs to the minimum fish flow, allowing the reservoir to rise to an unsafe state

-UCA24/UCA32 SPOG close command provided too late, reservoir drains [H2]

-Controller thinks reservoir level is higher than the gate sill (gauge failure or delay)

-Controller thinks inflow is high (inflow forecast error) so keeps reservoir open to pre-spill for a storm that never comes, resulting in the reservoir being drained to the sill

-Operator is responding to issues at another site and overlooks the fact that the CMS reservoir is draining to the sill

-MICROWAVE/Sat links fail, by the time site staff arrive to close gate, reservoir is below gate sill

-UCA33 LLO Open command not provided when SPOGs closed [H1, H2, H3]

-Controller thinks SPOGs are open (SPOG position sensor failure, or relay failure) [H1, H2, H3]

-SPOGs fail closed, controller unaware [H1, H2, H3]

-UCA34 LLO Open command provided when SPOGs open [H1]

-Controller thinks inflows are very high (inflow forecast error) and releases an excess amount of water downstream than is necessary to control reservoir level [H1]

-UCA35 LLO Open command provided too late [H1, H2, H3]

-SPOGs failed, high inflows, operator mobilization to site takes longer than expected (traffic/personnel issues/timing) [H1, H2, H3]

-UCA36 LLO Open command provided too long, reservoir drains [H2]

-Controller expects very high inflows (inflow forecast error). Operator mobilizes to site and opens gate then leaves site. When inflows shown to be low and reservoir is draining, operator must mobilize to site to close gate (site access issues, personnel issues)

-UCA37 LLO Open command stopped too early, reservoir level increases [H1, H3]

-Controller thinks inflows are manageable with SPOGs so operator mobilizes to site to close LLO. SPOGs fail closed after operator leaves site resulting in increase in reservoir elevation.

-Downstream flooding at Squamish due to high tide and tributary flows, so controller decides to hold water back and reduces LLO flow to minimum fish flow. SPOGs closed for maintenance.

-Failure of LLO in closed position, SPOGs closed and/or out of service

UCA38: LLO Close command not provided when reservoir level approaching sill [H2]

-Operator thinks reservoir level higher than it is (sensor failure/delay)

-Damage to dam structure means reservoir must be drained to avoid potential internal erosion issues (earthquake/aging). Inflow falls to below the set LLO outflow and reservoir falls below sill, resulting in fish kill.

UCA39: LLO Close command provided when inflows and reservoir elevation high [H1, H3]

-Controller feels inflows are manageable with SPOGs and power releases so operator closes LLO. SPOGs/power releases then fail after operator leaves site and reservoir rises to unsafe level [H1, H2, H3]

UCA40: LLO close command provided too early [H1, H2, H3]

-Controller feels inflows are manageable with SPOGs and power releases so operator closes LLO. Inflows increase (inflow forecast error) and/or SPOGs/Powerhouse fail

-SPOGs out of service for maintenance. During the reverse lockout procedure when SPOGs are being brought back online, operator closes LLO too early (communication error with colleagues on site), resulting in fish kill

UCA41: LLO Close command provided too late [H2]

-SPOGs out of service. Controller thinks reservoir level is higher than it is (sensor failure or delay). Once controller realizes it is approaching sill, staff are mobilized to site to close the gate, but mobilization takes too long and reservoir drained to sill of LLO (traffic, personnel, timing)

-UCA42/UCA46 Turbine Open command not provided when inflows high and SPOGs out of service [H1, H3]

-Grid unavailable so turbine cannot be opened

-Price of power is negative, PSOSE and FVO need to keep generation to a minimum while they import power from out of province

-UCA43/UCA47 Turbine Open command provided but does not work, SPOGs out of service [H1, H3]

-Load rejection at powerhouse, wicket gate failure, etc.

-Failure of remote control, site inaccessibility (forest fire, landslide, washouts in Squamish valley)

-UCA44/UCC48 Turbine Open command provided too late, SPOGs out of service and inflow high [H1, H3]

-Remote control failed (MICROWAVE and satellite or issue within powerhouse), site accessibility is delayed due to poor weather conditions and traffic

-UCA45/UCC49 Turbine Open command provided too long, reservoir level falls below power intake sill

-Power shortages in lower mainland so PSOSE and FVO opt to prioritize generation, drawing reservoir to below sill of SPOG/LLO

-Controller unaware that the reservoir is low (sensor failure) runs powerhouse until reservoir falls below SPOG/LLO sill resulting in fish kill

-UCA50/UCA54 Turbine Close command not provided when reservoir levels low [H2]

-Energy shortage in lower mainland pushes controllers (PSOSE, ACC) to keep generating when reservoir elevation dropping to below LLO sill

-Controller unaware reservoir is low (sensor failure) so powerhouse is run until reservoir falls below SPOG/LLO sill resulting in fish kill

-UCA51/UCA55 Turbine closes when command not provided causing water hammer and penstock rupture [H2, H3]

-Load rejection, plugging/collapse of surge shaft resulting in water hammer that ruptures penstock resulting in damage and draining of the reservoir to intake sill

-UCA52/UCA56 Turbine Close command provided when reservoir levels high [H1, H3]

-Price of energy becomes negative, SPOG1 and SPOG2 out of service for maintenance and inflows high. Controllers choose to close turbines and use LLO for spill releases, resulting in high reservoir elevations when inflows exceed LLO capacity

-Price of energy becomes negative. SPOG1 and SPOG2 subsequently fail closed and reservoir rises to unsafe elevation when inflows exceed LLO capacity

UCA53/UCA57 Turbine Close command provided too late, reservoir level low [H2]

-Remote control of powerhouse fails, water dropping to below sill of LLO before personnel can access site to close wicket gates (traffic, site access restriction due to fire hazard, washouts or landslide) [H2]

Further details about specific components/failures:

Other considerations/scenarios for Turbines:

-Rough load zone operation leading to failure of the head cover(s), draining reservoir through powerhouse. Though, there does not appear to be a rough load zone specified for the Cheakamus units. [H2, H3]

-Runaway turbine if generation/grid links severed? [H2, H3]

Non-control related considerations:

-Earthquake causing settlement of earth dam and/or toppling of wing/saddle/concrete dams [H1, H3]

-Barrier slide failure (may or may not be earthquake induced) leading to buildup of material in Cheakamus Valley downstream from dam resulting in inability to pass water through system, eventual overtopping/breach

Causes of failures of recurring components from STPA analysis:

RTU's (2): (1) Controls SPOGs, (2) Collects sensor info from PLC and relays to microwave/sat links. Failure resulting in loss of remote gate control and loss of visibility.

-Power supply failure (backup batteries at end of service life)

-Microprocessor failure

-Lightning (one or both)

-Earthquake causing structural movement of RTU/wiring/etc.

PLC:

-Voltage fluctuations causing them to lock themselves out. Automatically reboot or require manual reboot.

-Backup batteries at end of service life, voltage too low for PLC

-May lose visibility in the event of grid failure since everything is coming to this PLC and being transferred to RTU/microwave/satellite

Gate position sensors (2): One rotary, one linear. Rotary converts angular to distance linearized with shape of gate, lookup table within sensor to determine opening. Linear is a straight rod at the trunnion, linearized in PLC. Transfers info to PLC.

-Power supply failures (grid or rodent activity)

-Failure of linear sensor in the event of ice storm possible (exposed to elements)

-Linear sensor is temperature sensitive, in hot weather it may appear gate is moving when it is not.

-PLC issues when one is way off from the other?

Reservoir level sensors (3) + staff gauge: PLC takes standard deviation between each one. If outside 4cm nominal difference, sensor omitted. Average of 2 or 3 taken. If two fail, PLC reads reservoir elevation as failed and passes that to RTU for control centre relay

-Linearity issues

-Temperature issues both high and low temperatures

-Must be rearranged every quarter – potential for issues in readings at end of quarter.

-If failed, site staff would have to go read staff gauge for accurate reservoir level reading

Site access:

-2-2.5 hours optimal

-In emergency, may be able to instruct power crews in Squamish on how to open gate or fly in (helicopter)

-Forest fires, earthquakes (highway collapse), traffic, barrier slide could all prevent access

-Overtopping of dam would surely prevent access (it would be flooded. Once that happens, nobody can access site).

Microwave failure:

-Earthquake

-Ice

-Can only be fixed in summer, very much weather dependant

Gates:

-Multiple motors

-Can be operated by power drill via gearbox

-One gate stiffened for overtopping flows, one isn't

-Overtravel limit – mechanical switch failure stops drum from turning when gate opened too high

-Backup gate drive motor

-Gearbox or drum failure would be catastrophic

    -misalignment of gears (earthquake)

    -lubrication issue (though would still work for a time)

-May be designed to open with a single hoist (need to check this)

Appendix C: Operating States Database for Cheakamus System

Table C1: Reservoir-level database components for Cheakamus System

ReservoirLevelID	ReservoirLevelTypeID	ReservoirLevelName	OperatingStateID	OperatingStateName	OperatingStateTypeID	ImpactTypeID	CausalFactorID	CausalFactorTypeID	CausalFactorName	MaxDate	MinDate
46	1	Gate Pier	220	NA	2	4	226	25	None	365	1
22	1	Main Earth Dam	104	None	2	4	227	25	None	365	1
18	4	Dam PLC	138	Functional	2	4	252	25	None	365	0
19	4	Powerhouse PLC	140	Functional	2	4	253	25	None	365	0
16	5	Dam RTU	142	Functional	2	4	254	25	None	365	0
17	5	Powerhouse RTU	144	Functional	2	4	255	25	None	365	0
21	8	Main Dam	106	None	2	4	264	25	None	365	0
27	11	Backup Diesel Generator	113	None	2	4	265	25	None	365	0
29	12	Dam Access	115	Typical access time	2	2	266	25	None	365	0
28	12	Powerhouse Access	117	Typical access time	2	2	267	25	None	365	0
30	23	Res El Sensor 1	218	Reading correct	2	4	268	25	None	365	0
31	23	Res El Sensor 2	212	Reading correct	2	4	269	25	None	365	0
47	23	Res El Sensor 3	209	Reading correct	2	4	270	25	None	365	0
37	23	SPOG1Position_L	126	Reading correct	2	4	271	25	None	365	0
39	23	SPOG2Position_L	132	Reading correct	2	4	272	25	None	365	0
38	23	SPOG2Position_R	135	Reading correct	2	4	273	25	None	365	0
36	23	SPOG1Position_R	221	Reading normal	2	4	274	25	None	365	0
41	25	Power tunnel	175	None	2	4	275	25	None	365	0
42	26	Penstock	177	Normal operation	2	4	276	25	None	365	0
44	27	CMS Grid	187	Normal Operation	2	4	277	25	None	365	0
43	27	Rainbow Grid	189	Normal Operation	2	4	278	25	None	365	0
45	28	CMS Inflow Forecast	192	Inflow forecast normal	2	4	279	25	None	365	0
48	29	Site Staff Availability	214	Staff available	2	4	280	25	None	365	0

ReservoirLevelID	ReservoirLevelTypeID	ReservoirLevelName	OperatingStateID	OperatingStateName	OperatingStateTypeID	ImpactTypeID	CausalFactorID	CausalFactorTypeID	CausalFactorName	MaxDate	MinDate
18	4	Dam PLC	139	PLC offline	1	1	64	10	Voltage fluctuation	365	0
18	4	Dam PLC	139	PLC offline	1	1	65	1	Earthquake	365	0
16	5	Dam RTU	143	Offline	1	1	66	11	Lightning	274	120
17	5	Powerhouse RTU	145	Offline	1	1	67	11	Lightning	274	120
16	5	Dam RTU	143	Offline	1	1	68	1	Earthquake	365	0
17	5	Powerhouse RTU	145	Offline	1	1	69	1	Earthquake	365	0
21	8	Main Dam	107	Cracking of concrete	1	7	72	1	Earthquake	365	0
27	11	Backup Diesel Generator	114	Generator fails, no power	1	1	82	1	Earthquake	365	0
27	11	Backup Diesel Generator	114	Generator fails, no power	1	1	83	1	Earthquake	365	0
27	11	Backup Diesel Generator	114	Generator fails, no power	1	1	84	9	Aging	365	0
27	11	Backup Diesel Generator	114	Generator fails, no power	1	1	85	2	Lack of maintenance	365	0
29	12	Dam Access	116	Access dangerous, delayed or not possible	6	2	86	14	Traffic/traffic indicent	365	0
29	12	Dam Access	116	Access dangerous, delayed or not possible	6	2	87	4	Excessive rainfall causes road washout	365	0
29	12	Dam Access	116	Access dangerous, delayed or not possible	6	2	88	1	Earthquake	365	0
29	12	Dam Access	116	Access dangerous, delayed or not possible	6	2	89	16	Forest fire resulting in evacuation	273	181
19	4	Powerhouse PLC	141	PLC offline	1	1	94	10	Voltage fluctuation	365	0
19	4	Powerhouse PLC	141	PLC offline	1	1	95	1	Earthquake	365	0
41	25	Power tunnel	176	Power tunnel collapse	1	5	111	1	Earthquake	365	0
42	26	Penstock	178	Penstock rupture	1	9	114	1	Earthquake	365	0
28	12	Powerhouse Access	118	Access dangerous, delayed or not possible	6	2	152	14	Traffic/traffic indicent	365	0
28	12	Powerhouse Access	118	Access dangerous, delayed or not possible	6	2	153	4	Excessive rainfall causes road washout	365	0
28	12	Powerhouse Access	118	Access dangerous, delayed or not possible	6	2	154	1	Earthquake	365	0
28	12	Powerhouse Access	118	Access dangerous, delayed or not possible	6	2	155	16	Forest fire resulting in evacuation	365	0
44	27	CMS Grid	188	Grid failure	1	1	156	5	Ice storm	59	0
44	27	CMS Grid	188	Grid failure	1	1	157	7	Wind storm	365	0
44	27	CMS Grid	188	Grid failure	1	1	158	16	Forest fire destroys infrastructure	273	120
44	27	CMS Grid	188	Grid failure	1	1	159	11	Lightning destroys infrastructure	273	120

ReservoirLevelID	ReservoirLevelTypeID	ReservoirLevelName	OperatingStateID	OperatingStateName	OperatingStateTypeID	ImpactTypeID	CausalFactorID	CausalFactorTypeID	CausalFactorName	MaxDate	MinDate
43	27	Rainbow Grid	190	Grid failure	1	1	160	5	Ice storm	59	0
43	27	Rainbow Grid	190	Grid failure	1	1	161	7	Wind storm	365	0
43	27	Rainbow Grid	190	Grid failure	1	1	162	16	Forest fire destroys infrastructure	273	120
43	27	Rainbow Grid	190	Grid failure	1	1	163	11	Lightning destroys infrastructure	273	120
46	1	Gate Pier	195	Failure of gate pier	1	1	164	1	Earthquake	365	0
47	23	Res El Sensor 3	210	No Reading	1	1	180	1	Earthquake	365	0
47	23	Res El Sensor 3	210	No Reading	1	1	181	12	Rodent activity causes short in wiring	365	0
47	23	Res El Sensor 3	210	No Reading	1	1	182	2	Lack of maintenance	365	0
47	23	Res El Sensor 3	210	No Reading	1	1	183	9	Aging	365	0
47	23	Res El Sensor 3	208	Wrong Reading	8	3	184	2	Failed to recalibrate seasonally	365	0
47	23	Res El Sensor 3	208	Wrong Reading	8	3	185	17	High or low temps result in decalibration	365	0
37	23	SPOG1Position_L	128	Wrong Reading	8	3	186	5	Ice buildup	59	0
37	23	SPOG1Position_L	128	Wrong Reading	8	3	189	2	Lack of maintenance	365	0
37	23	SPOG1Position_L	128	Wrong Reading	8	3	190	1	Earthquake	365	0
39	23	SPOG2Position_L	134	Wrong Reading	8	3	191	5	Ice buildup causes sensor to decalibrate	59	0
39	23	SPOG2Position_L	134	Wrong Reading	8	3	192	2	Lack of maintenance	365	0
39	23	SPOG2Position_L	134	Wrong Reading	8	3	193	1	Earthquake	365	0
36	23	SPOG1Position_R	131	Wrong Reading	8	3	195	2	Lack of maintenance, sensor deteriorates	365	0
36	23	SPOG1Position_R	131	Wrong Reading	8	3	196	1	Earthquake	365	0
36	23	SPOG1Position_R	131	Wrong Reading	8	3	197	12	Rodent activity	365	0
38	23	SPOG2Position_R	137	Wrong Reading	8	3	198	2	Lack of maintenance	365	0
38	23	SPOG2Position_R	137	Wrong Reading	8	3	199	1	Earthquake	365	0
38	23	SPOG2Position_R	137	Wrong Reading	8	3	200	12	Rodent activity	365	0
31	23	Res El Sensor 2	211	Wrong Reading	8	3	204	2	Failed to recalibrate seasonally	365	0
31	23	Res El Sensor 2	211	Wrong Reading	8	3	205	17	High or low temps result in decalibration	365	0
31	23	Res El Sensor 2	213	No Reading	1	1	206	1	Earthquake	365	0
31	23	Res El Sensor 2	213	No Reading	1	1	207	12	Rodent activity	365	0

ReservoirLevelID	ReservoirLevelTypeID	ReservoirLevelName	OperatingStateID	OperatingStateName	OperatingStateTypeID	ImpactTypeID	CausalFactorID	CausalFactorTypeID	CausalFactorName	MaxDate	MinDate
31	23	Res El Sensor 2	213	No Reading	1	1	208	2	Lack of maintenance	365	0
31	23	Res El Sensor 2	213	No Reading	1	1	209	9	Aging	365	0
30	23	Res El Sensor 1	217	Wrong Reading	8	3	210	2	Failed to recalibrate seasonally	365	0
30	23	Res El Sensor 1	217	Wrong Reading	8	3	211	17	High or low temps result in decalibration	365	0
30	23	Res El Sensor 1	219	No Reading	1	1	212	1	Earthquake	365	0
30	23	Res El Sensor 1	219	No Reading	1	1	213	12	Rodent activity	365	0
30	23	Res El Sensor 1	219	No Reading	1	1	214	2	Lack of maintenance	365	0
30	23	Res El Sensor 1	219	No Reading	1	1	215	9	Aging	365	0
45	28	CMS Inflow Forecast	194	Inflow forecasting error	8	3	216	20	Operator fatigue	365	1
45	28	CMS Inflow Forecast	194	Inflow forecasting error	8	3	217	21	Uncertainty	365	1
48	29	Site Staff Availability	215	Staff unavailable	6	2	218	22	Weekend or evening	365	1
48	29	Site Staff Availability	215	Staff unavailable	6	2	219	23	Staff are busy and unable to access site	365	1

Table C2: Component-Level database components for Cheakamus System

ReservoirLevelID	ReservoirLevelTypeID	ReservoirLevelName	ComponentLevelID	ComponentLevelTypeID	ComponentLevelName	OperatingStateID	OperatingStateName	OperatingStateTypeID	ImpactTypeID	CausalFactorTypeID	CausalFactorName	Min Date	Max Date
13	2	Gate 1	26	1	Gate Hoist 1	183	Normal	2	4	25	None	0	365
13	2	Gate 1	28	2	Skinplate	23	Normal	2	4	25	None	0	365
13	2	Gate 1	31	5	Gearbox	35	Normal	2	4	25	None	0	365
13	2	Gate 1	32	10	Motor	36	Normal	2	4	25	None	0	365
13	2	Gate 1	33	11	Structural Supports	37	Normal	2	4	25	None	0	365
13	2	Gate 1	34	12	Hoist Gate Connection 1	38	Normal	2	4	25	None	0	365
13	2	Gate 1	43	16	Thrustor Brake	69	Normal	2	4	25	None	0	365
13	2	Gate 1	55	10	Backup Motor	165	Normal	2	4	25	None	0	365
13	2	Gate 1	57	19	Gate 1 Opening	179	Normal	2	4	25	None	0	365
14	2	Gate 2	16	1	Gate Hoist 2	48	Normal	2	4	25	None	0	365
14	2	Gate 2	18	2	Skinplate	44	Normal	2	4	25	None	0	365
14	2	Gate 2	21	5	Gearbox	57	Normal	2	4	25	None	0	365
14	2	Gate 2	22	10	Motor	58	Normal	2	4	25	None	0	365
14	2	Gate 2	23	11	Structural Supports	76	Normal	2	4	25	None	0	365
14	2	Gate 2	24	12	Hoist Gate Connection 2	65	Normal	2	4	25	None	0	365
14	2	Gate 2	44	16	Thrustor Brake	74	Normal	2	4	25	None	0	365
14	2	Gate 2	56	10	Backup Motor	167	Normal	2	4	25	None	0	365
14	2	Gate 2	58	19	Gate 2 Opening	181	Normal	2	4	25	None	0	365
8	3	Turbine 1	36	13	Head Cover	80	Normal	2	4	25	None	0	365
8	3	Turbine 1	37	14	Wicket Gates	83	Normal	2	4	25	None	0	365
8	3	Turbine 1	38	15	Generator	85	Normal	2	4	25	None	0	365
10	3	Turbine 2	39	13	Head Cover	87	Normal	2	4	25	None	0	365
10	3	Turbine 2	40	14	Wicket Gates	90	Normal	2	4	25	None	0	365
10	3	Turbine 2	41	15	Generator	92	Normal	2	4	25	None	0	365
15	7	Low Level Outlet	46	1	Hoist	146	Normal	2	4	25	None	0	365
15	7	Low Level Outlet	47	2	Skinplate	148	Normal	2	4	25	None	0	365

ReservoirLevelID	ReservoirLevelTypeID	ReservoirLevelName	ComponentLevelID	ComponentLevelTypeID	ComponentLevelName	OperatingStateID	OperatingStateName	OperatingStateTypeID	ImpactTypeID	CausalFactorTypeID	CausalFactorName	Min Date	Max Date
15	7	Low Level Outlet	48	10	Motor	150	Normal	2	4	25	None	0	365
15	7	Low Level Outlet	49	11	Support	152	Normal	2	4	25	None	0	365
15	7	Low Level Outlet	50	12	Hoist Gate Connection	155	Normal	2	4	25	None	0	365
15	7	Low Level Outlet	51	12	Thrustor Brake	157	Normal	2	4	25	None	0	365
15	7	Low Level Outlet	54	5	Gearbox	160	Normal	2	4	25	None	0	365
13	2	Gate 1	26	1	Gate Hoist 1	20	Steel yields, hoists fail, gate fails closed	3	1	5	Ice force on gate	1	69
13	2	Gate 1	26	1	Gate Hoist 1	20	Steel yields, hoists fail, gate fails closed	3	1	3	Debris force on gate	120	274
14	2	Gate 2	16	1	Gate Hoist 2	94	Steel yields, hoists fail, gate fails closed	3	1	5	Ice force on gate	0	69
14	2	Gate 2	16	1	Gate Hoist 2	94	Steel yields, hoists fail, gate fails closed	3	1	3	Debris force on gate	120	274
13	2	Gate 1	28	2	Skinplate	24	Steel yields	4	1	3	Debris force	120	274
13	2	Gate 1	28	2	Skinplate	24	Steel yields	4	1	2	Lack of maintenance	0	365
14	2	Gate 2	18	2	Skinplate	95	Steel yields	4	1	3	Debris force on gate	120	274
14	2	Gate 2	18	2	Skinplate	95	Steel yields	4	1	2	Lack of maintenance	0	365
13	2	Gate 1	31	5	Gearbox	34	Gearbox stripped	4	1	1	Movement of gears	0	365
13	2	Gate 1	31	5	Gearbox	34	Gearbox stripped	4	1	2	Lack of maintenance	0	365
14	2	Gate 2	21	5	Gearbox	96	Gearbox stripped	4	1	1	Movement of gears	0	365
14	2	Gate 2	21	5	Gearbox	96	Gearbox stripped	4	1	2	Lack of maintenance	0	365
13	2	Gate 1	26	1	Gate Hoist 1	20	Steel yields, hoists fail, gate fails closed	3	1	8	Both motors engage resulting in overforce	0	365
13	2	Gate 1	26	1	Gate Hoist 1	20	Steel yields, hoists fail, gate fails closed	3	1	8	Overforce alarm fails	0	365
14	2	Gate 2	16	1	Gate Hoist 2	94	Steel yields, hoists fail, gate fails closed	3	1	8	Both motors engage resulting in overforce	0	365
14	2	Gate 2	16	1	Gate Hoist 2	94	Steel yields, hoists fail, gate fails closed	3	1	8	Overforce alarm fails	0	365
13	2	Gate 1	32	10	Motor	40	Motor Failure	4	1	2	Lack of maintenance	0	365
13	2	Gate 1	32	10	Motor	40	Motor Failure	4	1	1	Earthquake	0	365
13	2	Gate 1	32	10	Motor	40	Motor Failure	4	1	9	Old motor	1	364
13	2	Gate 1	55	10	Backup Motor	166	Motor Failure	4	1	2	Lack of maintenance	0	365
13	2	Gate 1	55	10	Backup Motor	166	Motor Failure	4	1	1	Earthquake	0	365
13	2	Gate 1	55	10	Backup Motor	166	Motor Failure	4	1	9	Old motor	0	365

ReservoirLevelID	ReservoirLevelTypeID	ReservoirLevelName	ComponentLevelID	ComponentLevelTypeID	ComponentLevelName	OperatingStateID	OperatingStateName	OperatingStateTypeID	ImpactTypeID	CausalFactorTypeID	CausalFactorName	Min Date	Max Date
14	2	Gate 2	22	10	Motor	97	Motor Failure	4	1	2	Lack of maintenance	0	365
14	2	Gate 2	22	10	Motor	97	Motor Failure	4	1	1	Earthquake	0	365
14	2	Gate 2	22	10	Motor	97	Motor Failure	4	1	9	Old motor	0	365
14	2	Gate 2	56	10	Backup Motor	168	Motor Failure	4	1	2	Lack of maintenance	0	365
14	2	Gate 2	56	10	Backup Motor	168	Motor Failure	4	1	1	Earthquake	0	365
14	2	Gate 2	56	10	Backup Motor	168	Motor Failure	4	1	9	Old motor	0	365
13	2	Gate 1	33	11	Structural Supports	41	Supports deform and gate collapses	5	1	1	Earthquake	0	365
13	2	Gate 1	33	11	Structural Supports	41	Supports deform and gate collapses	5	1	8	Overforce alarm fails	0	365
14	2	Gate 2	23	11	Structural Supports	98	Supports deform and gate collapses	5	1	1	Earthquake	0	365
14	2	Gate 2	23	11	Structural Supports	98	Supports deform and gate collapses	5	1	8	Overforce alarm fails	0	365
13	2	Gate 1	33	11	Structural Supports	42	Supports deform and gate becomes immoveable	4	1	1	Earthquake	0	365
14	2	Gate 2	23	11	Structural Supports	99	Supports deform and gate becomes immoveable	4	1	1	Earthquake	0	365
13	2	Gate 1	34	12	Hoist Gate Connection 1	43	Gate connection snaps	3	1	9	Aging	0	365
14	2	Gate 2	24	12	Hoist Gate Connection 2	100	Gate connection snaps	3	1	9	Aging	0	365
13	2	Gate 1	43	16	Thrustor Brake	70	Brake fails, gate closes	3	1	9	Aging	0	365
13	2	Gate 1	43	16	Thrustor Brake	70	Brake fails, gate closes	3	1	8	Feedback failure	0	365
14	2	Gate 2	44	16	Thrustor Brake	101	Brake fails, gate closes	3	1	9	Aging	0	365
14	2	Gate 2	44	16	Thrustor Brake	101	Brake fails, gate closes	3	1	8	Feedback failure	0	365
13	2	Gate 1	57	19	Gate 1 Opening	180	Opening is blocked	2	5	3	Debris accumulates at gate opening	120	274
14	2	Gate 2	58	19	Gate 2 Opening	182	Opening is blocked	2	5	3	Debris accumulates at gate opening	120	274
8	3	Turbine 1	36	13	Head Cover	81	Bolt fatigue, head cover failure	5	9	2	Lack of maintenance	0	365
8	3	Turbine 1	37	14	Wicket Gates	82	Wicket gates fail closed	1	1	2	Lack of maintenance	0	365
10	3	Turbine 2	39	13	Head Cover	88	Bolt fatigue, head cover failure	5	9	2	Lack of maintenance	0	365
15	7	Low Level Outlet	48	10	Motor	151	Motor Failure	4	1	2	Lack of maintenance	0	365
15	7	Low Level Outlet	48	10	Motor	151	Motor Failure	4	1	1	Earthquake	0	365
15	7	Low Level Outlet	48	10	Motor	151	Motor Failure	4	1	9	Old motor	0	365
15	7	Low Level Outlet	49	11	Support	184	Supports deform and gate collapses	5	1	1	Earthquake	0	365

ReservoirLevelID	ReservoirLevelTypeID	ReservoirLevelName	ComponentLevelID	ComponentLevelTypeID	ComponentLevelName	OperatingStateID	OperatingStateName	OperatingStateTypeID	ImpactTypeID	CausalFactorTypeID	CausalFactorName	Min Date	Max Date
15	7	Low Level Outlet	49	11	Support	184	Supports deform and gate collapses	5	1	8	Feedback failure	0	365
15	7	Low Level Outlet	49	11	Support	185	Supports deform and gate becomes immoveable	4	1	1	Earthquake	0	365
15	7	Low Level Outlet	50	12	Hoist Gate Connection	156	Gate connection snaps	3	1	9	Aging	0	365
15	7	Low Level Outlet	51	12	Thrustor Brake	158	Brake fails, gate closes	3	1	9	Aging	0	365
15	7	Low Level Outlet	51	12	Thrustor Brake	158	Brake fails, gate closes	3	1	8	Feedback failure	0	365
15	7	Low Level Outlet	54	5	Gearbox	159	Gearbox stripped	4	1	1	Earthquake	0	365
15	7	Low Level Outlet	54	5	Gearbox	159	Gearbox stripped	4	1	2	Lack of maintenance	0	365
15	7	Low Level Outlet	47	2	Skinplate	149	Steel yields	4	1	2	Lack of maintenance	0	365
15	7	Low Level Outlet	47	2	Skinplate	149	Steel yields	4	1	3	Debris force on gate	0	365
13	2	Gate 1	26	1	Gate Hoist 1	20	Steel yields, hoists fail, gate fails closed	3	1	2	Lack of maintenance	0	365
14	2	Gate 2	16	1	Gate Hoist 2	94	Steel yields, hoists fail, gate fails closed	3	1	2	Lack of maintenance	0	365
15	7	Low Level Outlet	46	1	Hoist	186	Steel yields, hoists fail, gate fails closed	3	1	8	Both motors engage resulting in overforce	0	365
15	7	Low Level Outlet	46	1	Hoist	186	Steel yields, hoists fail, gate fails closed	3	1	8	Feedback failure	0	365
15	7	Low Level Outlet	46	1	Hoist	186	Steel yields, hoists fail, gate fails closed	3	1	2	Lack of maintenance	0	365
15	7	Low Level Outlet	46	1	Hoist	186	Steel yields, hoists fail, gate fails closed	3	1	5	Ice force on gate	0	365
15	7	Low Level Outlet	46	1	Hoist	186	Steel yields, hoists fail, gate fails closed	3	1	3	Debris force on gate	0	365
8	3	Turbine 1	38	15	Generator	86	Load Rejection	1	1	2	Lack of maintenance	0	365
10	3	Turbine 2	41	15	Generator	93	Load Rejection	1	1	2	Lack of maintenance	0	365
10	3	Turbine 2	40	14	Wicket Gates	216	Wicket gates fail closed	1	1	2	Lack of maintenance	0	365

Appendix D: Operating States Database for Simplified System

Table D1: Reservoir-Level database components for Simplified System

Ind	Reservoir LevelId	ReservoirLevel TypeId	ReservoirLevel Name	Operating StateId	OperatingStateName	Operating StateTypeId	Impact TypeId	Min	Max	Avg	UnitId	Causal FactorId	CausalFactor TypeId	CausalFactor Name	Max Date	Min Date
18_1	18	4	PLC/RTU	139	PLC offline	1	1	1	24	6	2	64	10	Voltage Fluctuation	365	0
18_2	18	4	PLC/RTU	139	PLC offline	1	1	1	24	6	2	65	1	Earthquake	365	0
29_1	29	12	Dam Access	116	Access dangerous, delayed or not possible	6	2	4	48	12	1	86	14	Traffic	365	0
29_2	29	12	Dam Access	116	Access dangerous, delayed or not possible	6	2	4	48	12	1	88	1	Earthquake	365	0
29_3	29	12	Dam Access	116	Access dangerous, delayed or not possible	6	2	4	48	12	1	89	16	Forest Fire	273	181
42_1	42	26	Penstock	178	Penstock rupture	1	9	60	365	90	1	114	1	Earthquake	365	0
44_1	44	27	Grid	188	Grid failure	1	1	0.04	7	0.167	1	157	7	Wind storm	365	0
44_2	44	27	Grid	188	Grid failure	1	1	0.04	7	0.167	1	158	16	Forest Fire	273	120
30_1	30	23	Reservoir Elevation Sensor 1	217	Wrong Reading	8	3	10	100	25	11	211	17	Temperature	365	0
30_2	30	23	Reservoir Elevation Sensor 1	219	No Reading	1	1	0.167	5	1	1	212	1	Earthquake	365	0
30_3	30	23	Reservoir Elevation Sensor 1	219	No Reading	1	1	0.167	5	1	1	214	2	Lack of Maintenance	365	0
45_1	45	28	CMS Inflow Forecast	194	Inflow forecasting error	8	3	-3	3	0	8	217	21	Uncertainty	365	1
48_1	48	29	Site Staff Availability	215	Staff unavailable	6	2	1	24	4	2	218	22	Timing	365	1
48_2	48	29	Site Staff Availability	215	Staff unavailable	6	2	1	24	4	2	219	23	Timing	365	1
18_3	18	4	PLCRTU	138	Functional	2	4	0	0	0	1	252	25	None	365	0
29_4	29	12	Dam Access	115	Typical access time	2	2	2	4	2.5	2	266	25	None	365	1
30_4	30	23	Reservoir Elevation Sensor 1	218	Reading correct	2	4	0	0	0	4	268	25	None	365	0
42_2	42	26	Penstock	177	Normal operation	2	4	0	0	0	1	276	25	None	365	0
44_3	44	27	Grid	187	Normal Operation	2	4	0	0	0	1	277	25	None	365	1
45_2	45	28	CMS Inflow Forecast	192	Inflow forecast normal	2	4	0	0	0	11	279	25	None	365	0
48_3	48	29	Site Staff Availability	214	Staff available	2	4	0	0	0	1	280	25	None	365	0
18_4	18	4	PLC/RTU	139	PLC offline	1	1	1	24	6	2	292	11	Lightning	273	120
30_5	30	23	Reservoir Elevation Sensor 1	219	No Reading	1	1	0.167	5	1	1	213	12	Rodent Activity	365	0
44_4	44	27	Grid	188	Grid failure	1	1	0.04	7	0.16	1	159	11	Lightning	273	120

Table D2: Component-Level database components for Simplified System

Identifier	Reservoir LevelID	Reservoir Level TypeID	Reservoir Level Name	Component LevelID	Component LevelTypeID	Component LevelName	Operating StateID	Operating StateName	Operating StateTypeID	Impact TypeID	Min	Max	Avg	Causal FactorID	Causal Factor TypeID	CausalFactor Name	Min Date	Max Date
836_1	8	3	Turbine 1	36	13	Head Cover	81	Bolt fatigue, reservoir drained through turbine hole	5	9	365	730	365	117	2	Maintenance	1	365
838_1	8	3	Turbine 1	38	15	Generator	86	Load Rejection	1	1	0.1	7	0.25	201	2	Maintenance	1	365
836_2	8	3	Turbine 1	36	13	Head Cover	80	Normal	2	4	0	0	0	246	25	None	1	365
838_2	8	3	Turbine 1	38	15	Generator	85	Normal	2	4	0	0	0	248	25	None	1	365
1359_1	13	2	Gate 1	59	19	Gate opening	222	Normal	2	4	0	0	0	281	25	None	1	365
1359_2	13	2	Gate 1	59	19	Gate opening	223	Gate is blocked by debris	1	5	10	80	20	282	3	Debris	90	334
1360_1	13	2	Gate 1	60	5	Components failing in place	225	Components of the gate fail causing it to remain in place	4	1	0.5	120	7	283	1	Earthquake	1	365
1360_2	13	2	Gate 1	60	5	Components failing in place	225	Components of the gate fail causing it to remain in place	4	1	0.5	120	7	284	2	Maintenance	1	365
1361_1	13	2	Gate 1	61	11	Components failing open	226	Normal	2	4	0	0	0	285	25	Normal	1	365
1361_2	13	2	Gate 1	61	11	Components failing open	227	Components of the gate collapse and water is released	5	1	210	730	240	286	1	Earthquake	1	365
1361_3	13	2	Gate 1	61	11	Components failing open	227	Components of the gate collapse and water is released	5	1	210	730	240	287	8	Feedback Failure	1	365
1362_1	13	2	Gate 1	62	1	Components failing closed	228	Normal	2	4	0	0	0	288	25	Normal	1	365
1362_2	13	2	Gate 1	62	1	Components failing closed	229	Components of the gate fail causing it to close	3	1	1	120	20	289	9	Aging	1	365
1362_3	13	2	Gate 1	62	1	Components failing closed	229	Components of the gate fail causing it to close	3	1	1	120	20	290	5	Ice	1	59
1362_4	13	2	Gate 1	62	1	Components failing closed	229	Components of the gate fail causing it to close	3	1	1	120	20	291	8	Feedback Failure	1	365
1360_3	13	2	Gate 1	60	5	Components failing in place	224	Normal	2	4	0	0	0	294	25	None	1	365

Appendix E: Simulation Script Organization and Discussion

The general description of the steps within the simulation model code is shown in Figure . The complete code is presented in Appendix E. The following sub-sections provide additional information about the equations used in the simulation model.

The first section of the code is entitled "1. Initialization", where required packages are imported and data files are read in to be utilized within the code. The simulation model requires several supporting files, including input CSV's containing information such as synthetic inflows, fish flow requirements, database data, baseline operating conditions (no failure), and rating curves for the spillway. In addition to this, several Python packages must be installed prior to running. These are listed below:

- numpy
- pandas
- time
- datetime
- sys
- argparse
- os
- random

Many of the aforementioned packages are available from an open source Anaconda3 installation at https://www.anaconda.com/distribution/. Most of the packages can be installed easily using *conda install* in the command prompt. In addition to these packages, the *sdpy* project must also be imported, as well as scenarios.py. These files and all necessary input files are available in the electronic appendix in the Dam_Safety_Model folder.

In the initialization section, arguments are also defined that allow the program to be called from the command prompt with a user-specified seed number and set the number of years/iterations (NYr) to be simulated. The seed number represents the scenario number

and for the simplified system it can vary between 0 and 552,960 (the total number of scenarios).

The following portion of the code is entitled “2. Generating Seeds”. The seed number is used to retrieve the scenario operating state identifiers as shown in the example scenario in the previous section. These operating state identifiers are used to extract the pertinent database information for the scenario of interest and convert the information into Monte-Carlo parameters for simulation. The “boolop” parameter is used to determine whether a seed should be randomized or not. It may be set equal to zero for script testing purposes, which will set all randomized components of the code to a single value – start dates are set to zero, start years are set sequentially from zero, average values are chosen for impacts, and all impacts occur on day 1 of the simulation. If “boolop” is set equal to one, a complete Monte-Carlo randomization of the inputs is performed. Starting dates from the inflow sequences are randomized using a start day (0-364) and a start year (0 to the number of synthetic inflow years simulated) – these can then be used to select the inflows for the simulation. The impacts for each operating state are randomized using the triangular distribution, with the minimum, maximum and average values as the inputs. Timing of impacts is also randomized, with impacts that have the same causal factor occurring on the same day except for maintenance and aging issues. The first impact begins on day 1 and subsequent impacts can occur on any day between day 1 and day $Tmax$. For the case study, $Tmax$ is set equal to the sum of the outage lengths generated for the iteration being considered. Truncating the maximum timing allows for the impacts of each operating state to be realized and, if possible, recovered from, during the simulation time frame. $Tmax$ is selected with the goal of increasing the number of “complete iterations”, where all events in a scenario both occur and affect one another. This will depend on the system being modelled and how flashy the reservoir is.

It is important to note that two runs were simulated for the case study (see details in Section 4.5.2). Some of the differences between functions within the two runs are also described throughout the remaining text in this section.

The next section of the simulation code is entitled “3. Initializing Supporting Functions and Arrays” sets up functions and arrays to be utilized within the simulation model. The supporting functions are not directly part of the system dynamics model but may be called by it many times during the simulation. The supporting functions are optimized, if

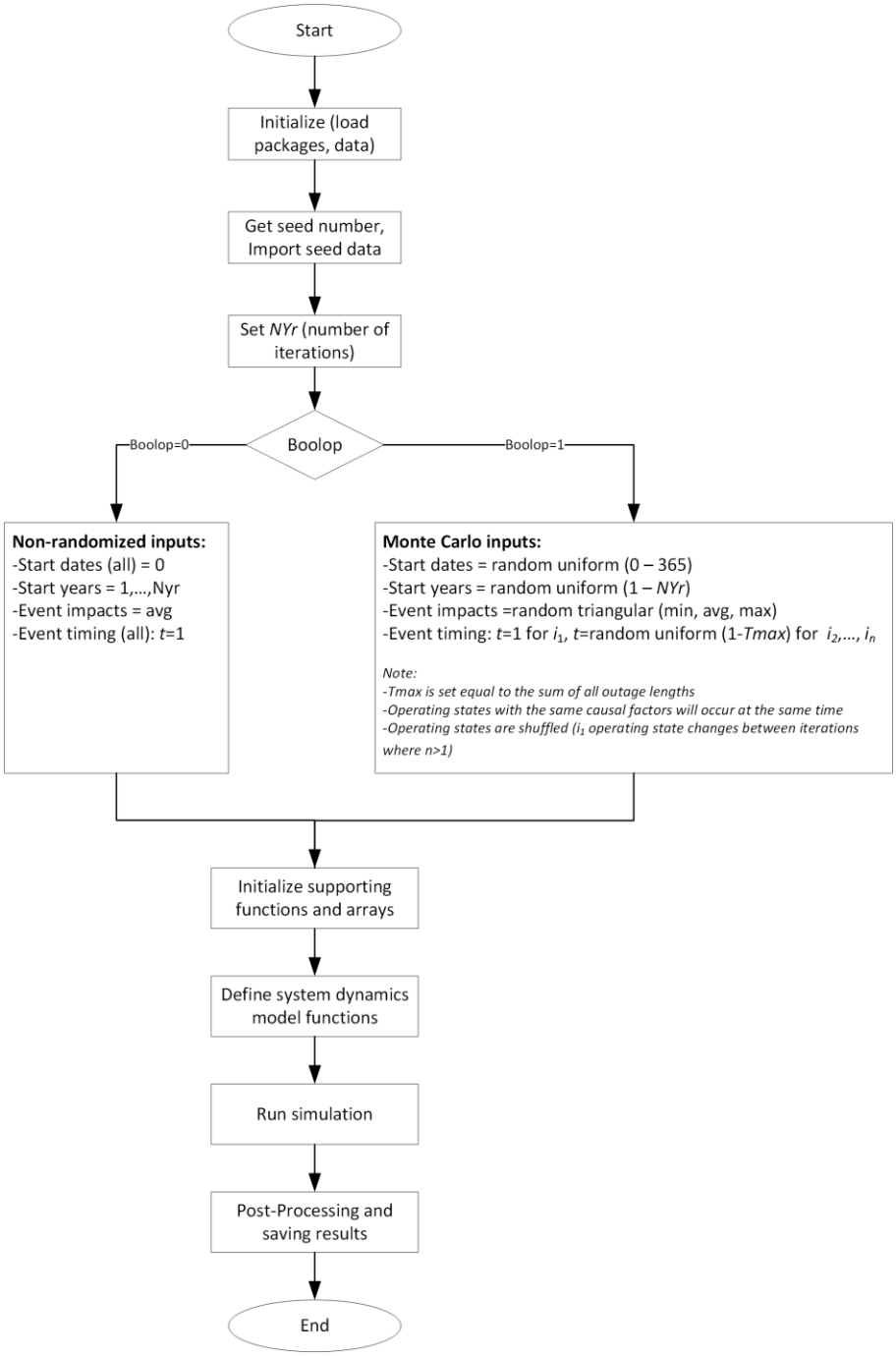


Figure E1: Simulation model process steps

possible, using “jit” – a “just-in-time” compiler that optimizes their performance. This ensures these functions, which are called many times in a given simulation, are executed as quickly as possible. The functions include:

- Stage-storage curve (SSC) which determines reservoir elevation *El* from storage *S* as in Equation 25, and it’s reverse (SSCRev) which finds the roots of Equation 25 to determine storage from elevation. These functions are located at line 724 and 731 of the code at the end of this Appendix.

$$El = -1.1201e-05\,S^2 + 0.032473\,S + 364.6572 \qquad (E.1)$$

- Stage-discharge curve for the free overflow sections (OTC), which is calculated using Equation 26, but is manipulated in the base case by increasing the spillway crest elevation by 2m and multiplying the result by 0.3 to represent a scaled down capacity of the free overflow structures in the base case. This was done to induce a larger number of dam failures for the proof-of-concept, since the spillway capacity of the real Cheakamus Dam is generally sufficient enough to prevent major consequences in even the most extreme scenarios. The code representing the two different overtopping curves for the base case and the dam safety improved case can be found on lines 734 and 2052, respectively, in the code at the end of this Appendix.

$$QOT = -35.75\,El^3 + 40896.27\,El^2 - 15593240.1\,El + 1981715583.1 \qquad (E.2)$$

- Maximum flow calculator for the gate (SPOGMaxFlow), which follows the piecewise Equation __ (see line 763 of the code at the end of this Appendix). The maximum flow through the gate is a function of the reservoir level RSE and the gate availability av, and was calculated in excel from the combined rating curve for two Cheakamus gates.

$$SPOGQ_{MAX}(RSE, av = 1)$$

$$= \begin{cases} 0 & if\ RSE < 367.28 \\ 19.1(RSE) - 7011.7 & if\ 367.28 \leq RSE < 367.5 \\ 37.3(RSE) - 13715.8 & if\ 367.5 \leq RSE < 367.8 \\ 49.7(RSE) - 18252 & if\ 367.8 \leq RSE < 369 \\ 2.15(RSE)^2 - 1496.34(RSE) + 258875.4 & if\ 369 \leq RSE < 381.6 \\ 861.1 + 728.9 & if\ RSE \geq 381.6 \end{cases}$$

$$SPOGQ_{MAX}(RSE, av = 0) = 0 \qquad (E.3)$$

- Maximum flow calculator for the turbine (fncTurbineMaxFlow), as per Equation E.4. This function computes the maximum flow through the turbine for a given reservoir level, and was calculated from the combined gross head-power-flow curves from the two Cheakamus units (see line 748 in the code at the end of this Appendix.).

$$TQ_{MAX}(RSE, av = 1)$$

$$= \begin{cases} 0 & if\ RSE < 363.05 \\ 13.98(RSE) - 5075.44 & if\ 363.05 \leq RSE < 365.05 \\ 18.02(RSE) - 6551.46 & if\ 365.05 \leq RSE < 367.05 \\ 65 & if\ RSE \geq 367.05 \end{cases}$$

$$TQ_{MAX}(RSE, av = 0) = 0 \qquad (E.4)$$

- A function to convert the spillway flow and reservoir elevation to gate instructions (GateInstr at line 782 in the code at the end of this Appendix) and a function to convert gate position and reservoir level to gate flow (GateFlowClac at line 828). These functions utilize a simple two-dimensional interpolation from the combined spillway gate rating curves provided by BC Hydro. The relationships between the maximum gate opening and discharge are shown in Appendix A.

- A function which finds the value of y_0 by linear interpolation using the two closest point pairs (x_1, y_1), (x_2, y_2) to a given x_0, following Equation E.5 (interpolate at line 814 in the code at the end of this Appendix):

$$y_0 = y_1 + (x_0 - x_1)\frac{y_2 - y_1}{x_2 - x_1} \qquad (E.5)$$

- A function to calculate the date reference number that is used to determine the time of year for reservoir level limits and minimum flow releases (dayrefs at line 923) – this simply uses the startday (0 to 365) and the time step to determine the day of the year.

- A function to get the minimum flow release for the upcoming days (getfishflow) which simply inserts the day-reference index into the fish flow array (line 902 in the code at the end of this Appendix)

- A function to generate an availability array for a component based on the total outage time (availarray at line 1033) – this simply converts an interger into an array of zeros and ones that represent whether a component is available or unavailable over a 14 day window from the current day – for example an input value of 8 would mean the component is unavailable for the next 8 days, and would produce an array [0,0,0,0,0,0,0,0,1,1,1,1,1,1].

- The main operations planning algorithm, which takes several key inputs (inflow forecast, reservoir elevation, day references, component availabilities and reservoir elevation limits) and determines the corresponding operating instructions for the system to ensure minimum flow releases are met and reservoir level restrictions are adhered to if possible (OpsPlan at line 960 in the code at the end of this Appendix). This follows a similar if-then-else type algorithm as presented in Figure 3-14, but with power flow releases added – see Figure . The algorithm begins by assuming the minimum fish flow is released and the remainder of the inflow is passed through the powerhouse (up to the maximum) for a 14-day window from the current date. The resultant reservoir levels are then checked, adjusted and re-checked to ensure

the operating instructions result in reservoir levels that are within the specified normal maximum (NMax) and minimum (NMin). To ensure enough water is available for the winter low-flow period, the normal minimum reservoir level was adjusted to El. 370 m for the months of November and December for the purposes of the modelling.

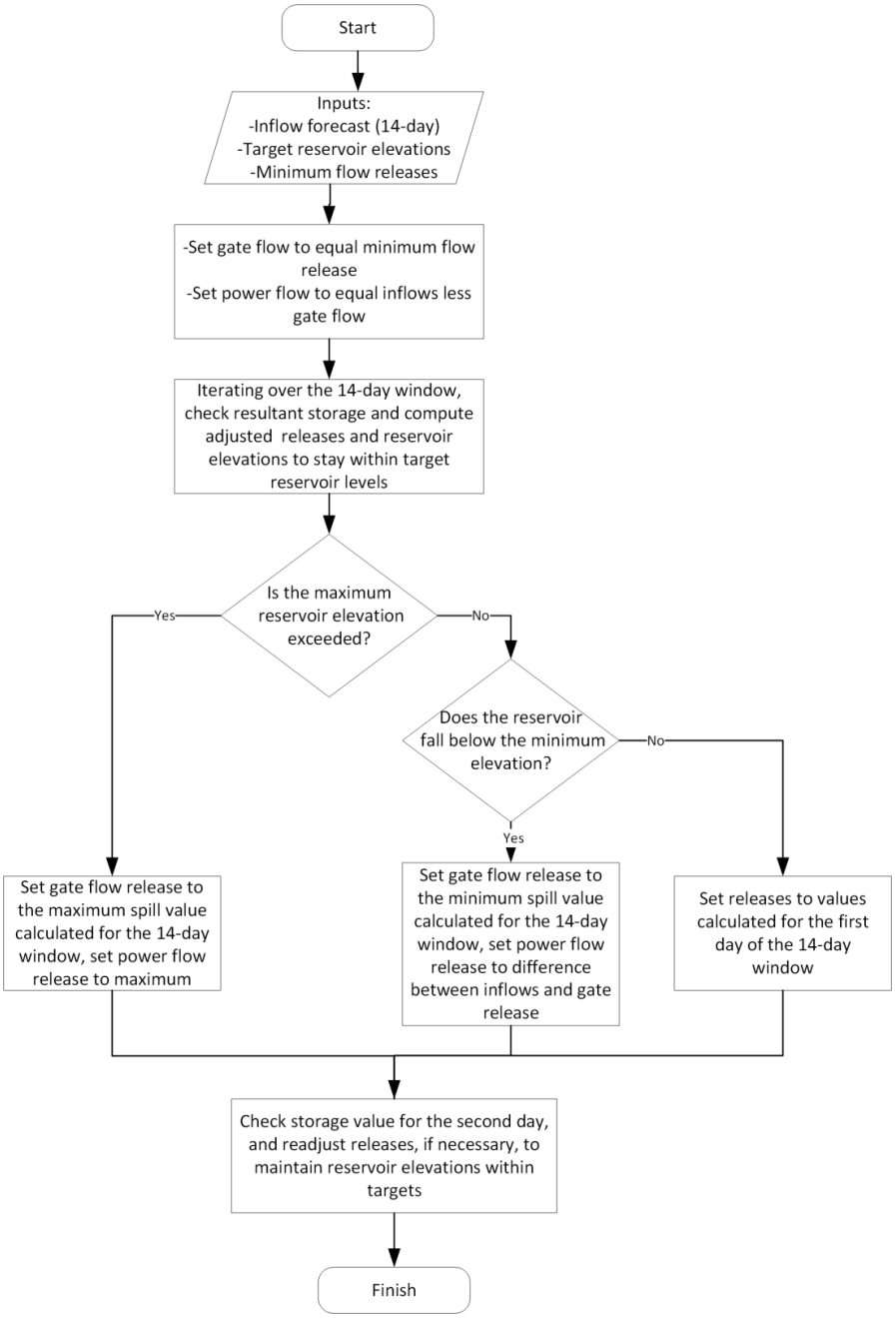


Figure E2: Operations Planning algorithm for simulation model

- A function to retrieve the normal operating maximum and minimum reservoir elevation corresponding to the reference day (GetNMax at line 1076 in the code at the end of this Appendix) – this simply inserts the day reference (day of the year) into an array containing the normal maximum and minimum levels for each day of the year.

The next section of code (4. Defining sdpy functions) contains the details of the system dynamics model, broken down into sectors. The functions for the hydraulic system state are shown in 4.1, the sensors in 4.2, the disturbances in 4.3, operations in 4.4, gate actuators in 4.5 and turbine actuators in 4.6. The output_saving function (line 1565) is used to store information from the model in memory for later post-processing. The model sectors are described in sub-sections 4.4.1.1 to 4.4.1.6.

Section 5 of the code (line 2005) is where the base case model is run. This section utilizes the features of the *sdpy* package to run the simulation for each of the iterations. Stocks must be re-set to their initial value following each iteration. Section 6 (line 2035) contains some re-initiation of key functions to reflect the changes in the dam safety improved case and also saves the results from the first run (base case) under different names for post-processing. Section 6 then re-runs the model with the same inputs as the base case for the dam safety improved case.

Section 7 of the script (line 2220) contains post-processing of the data, which involves the analysis of event dependency and saving of pertinent data from the scenario into ".npz" file formats. This format represents a compressed dictionary of arrays that are easily loaded into python for future analysis and plotting.

```
"""
SIMULATION SCRIPT

This code contains the necessary script to run the simplified system dynamics model,
with seed inputs as defined by the user, as well as NYr input to set the number of
iterations. Flush_period argument should remain at 1.

@author: Leanna King
"""

"""
1. INITIALIZATION

-Importing necessary directories
-Reading and organizing input files

"""

import time
t0=time.time()
from numba import njit, jit
#from scipy.interpolate import interp2d
#from scipy.interpolate import interp1d
import numpy as np
import pandas as pd
from datetime import datetime
import sdpy
import sys
import argparse
from scenarios import all_scenarios
import os


def datafile_path(filename):
    return os.path.join(os.path.dirname(sys.argv[0]), 'data', filename)

def RangeConstrainedParam(name, minvalue, maxvalue):
    def parse(s):
        n=int(s)
        if n < minvalue or n > maxvalue:
            raise ValueError()
        return n
    parse.__name__ = name
    return parse

maxseednum = len(all_scenarios)-1
parser = argparse.ArgumentParser(description='')
parser.add_argument('--NYr',
                    type=RangeConstrainedParam('NYr', 1, 10000),
                    default=5)
parser.add_argument('--seednum',
                    type=RangeConstrainedParam('seednum', 0, maxseednum),
                    default=301476) #813840 represents normal conditions
parser.add_argument('--flush_period',
                    type=RangeConstrainedParam('flush_period', 1, 10000),
                    default=1)
args = parser.parse_args()


```

```
60. model = sdpy.Sdmodel()
61.
62. NYr=args.NYr
63. seednum = args.seednum
64.
65. year=0
66. boolop=1 #Set to 1 for randomized, zero for non-randomized.
67. seedgen=1 #Set to 1 to generate seed, set to zero to read a previously generated
     seed file.
68.
69.
70. NormalRSEs=pd.read_csv(datafile_path(r'BaselineRSEs.csv'), header=0).values
71. NormalTBFs=pd.read_csv(datafile_path(r'BaselineTBFs.csv'), header=0).values
72. NormalSPOGs=pd.read_csv(datafile_path(r'BaselineSPOGs.csv'), header=0).values
73. NormalOTs=pd.read_csv(datafile_path(r'BaselineOTs.csv'), header=0).values
74. NormalTSs=NormalSPOGs
75.
76. #Inflow timeseries - SYNTHETIC - 10,374 years
77. Inflow_year=np.loadtxt(datafile_path("SyntheticInflow_Years.txt"), delimiter=","
    )
78. InflowJan1Start=np.zeros((365*2,10373))
79. for yr in range(10373):
80.     InflowJan1Start[0:365, yr]=Inflow_year[:,yr]
81.     InflowJan1Start[365:730,yr]=Inflow_year[:,yr+1]
82.
83.
84. """
85. 2. GENERATING SEEDS
86.
87. -The seeds set the randomized parameters for each NYr using a Monte-
    Carlo framework
88. -Seeds can be loaded from a previous output file if seedgen=0
89. -Or, create a seed from the seed number if seedgen=1
90. """
91.
92. ScenarioRL=datafile_path("S_RLAll-Inds-d.csv")
93. ScenarioCL=datafile_path("S_CLAll-Inds-d.csv")
94.
95. #print("Seednum: "+str(seednum))
96.
97. deltatmax=40 #set this based on system, or make it variable depending on number
    of adverse OS's
98.
99. if seedgen==0:
100.           Seeds=np.load(str("Outputs-"+str(seednum)+"-e.npz"))
101.
102.       if seedgen==1:
103.           import time
104.           import random
105.
106.           #Defining dictionaries
107.           #Reservoir Level components
108.           RLev={13: "G",
109.                   8: "T",
110.                   18: "PLR",
111.                   29: "ACC",
112.                   42: "PN",
113.                   44: "GD",
114.                   30: "SN",
115.                   48: "STF",
116.                   45: "IF"
```

```
                }

        #Impact Types
        Impacts={1: "o",
                 2: "d",
                 3: "e",
                 5: "bl",
                 4: "n",
                 9: "ur"
                }

        #OS Types
        OSTypes={1: "f",
                 2: "n",
                 3: "fc",
                 4: "fip",
                 5: "c",
                 6: "d",
                 8: "e"
                }

        #Component level components
        CLType={13: "HC",
                15: "GEN",
                19: "GO",
                5: "FIP",
                11: "FO",
                1: "FC",
               }

        CFType={1: "eq",
                2: "mt",
                3: "deb",
                4: "ra",
                5: "ice",
                7: "wnd",
                8: "fb",
                9: "age",
                10: "vf",
                11: "ltg",
                12: "rat",
                13: "dsg",
                14: "trf",
                15: "wsh",
                16: "fir",
                17: "tem",
                18: "wha",
                20: "ofa",
                21: "unc",
                22: "tim",
                23: "emr",
                25: "non",
                }

        CFDates={1: [0,365],
                 2: [0,365],
                 3: [90,334],
                 4: [0,365],
                 5: [0, 60],
                 7: [0,365],
                 8: [0,365],
```

```
                    9: [0,365],
                    10: [0,365],
                    11: [90, 334],
                    12: [0,365],
                    13: [0,365],
                    14: [0,365],
                    15: [0,365],
                    16: [151, 304],
                    17: [0,365],
                    18: [0,365],
                    20: [0,365],
                    21: [0,365],
                    22: [0,365],
                    23: [0,365],
                    25: [0,365],
                   }
        #    @jit
            def gen_avg(expected_avg, n, a, b, boolop):
                if boolop==1: #random
                    out=np.random.uniform(a, b, n)
                if boolop==0: #nonrandom
                    out=np.ones(n)*expected_avg
                return out
        #    @jit
            def dayrefs(Startdays, timestep, NYr):
                daynum=Startdays+int(timestep) #STARTDAYS CONTAINS 1-
    365 STARTING DAY REF FOR EACH INFLOW SEQUENCE
                dayref365=np.zeros((365,NYr))
                dayref365[0, :]=daynum
                for t in range(364):  #Converts vensim date into numbers 1-
    365 to represent dates in the model
                    for yr in range(NYr):
                        if dayref365[t, yr]+1<365:
                            dayref365[t+1, yr]=dayref365[t, yr]+1
                        else: dayref365[t+1, yr]=0
                return dayref365
        #    @jit
            def randintswitch(low, high, NYears ,boolop):
                if boolop==1: #randomly sets int between low and high
                    out=np.zeros(NYears)
                    for i in range(NYears):
                        out[i]=np.random.randint(low, high[i], 1)
                    return out
                if boolop==0: #defaults to low if non-randomized
                    return np.ones(NYears)*low
        #    @jit
            def randintswitch2(low, high, NYears ,boolop):
                if boolop==1: #randomly sets int between low and high
                    return np.random.randint(low, high, NYears)
                if boolop==0: #defaults to low if non-randomized
                    return np.ones(NYears)*low
        #    @jit
            def randintswitchCF(mindate, maxdate, refdates, lowest, highest,  bo
    olop):
                if boolop==1: #randomly sets int between low and high
                    if mindate!=0 or maxdate!=365:
                        clipped=np.clip(refdates, mindate, maxdate)
                        dates=np.random.randint(np.min(clipped), np.max(clipped)
    )
                        return np.where(refdates==dates)[0]
                    else:
```

```
235.                     return np.random.randint(lowest, highest)
236.                 if boolop==0: #defaults to low if non-randomized
237.                     return lowest
238.         #    @jit
239.             def randswitch(mini, avg, maxi, Nyears, boolop):
240.                 if boolop==1:
241.                     return np.random.triangular(mini, avg, maxi, Nyears) #return
    s triangular distributed variables
242.                 if boolop==0:
243.                     return np.ones(Nyears)*avg
244.
245.             name1=ScenarioRL
246.             name2=ScenarioCL
247.
248.             S_RL=pd.read_csv(name1)
249.             S_RL=S_RL.set_index("NewInd")
250.             S_CL=pd.read_csv(name2)
251.             S_CL=S_CL.set_index("NewInd") #setting index to the formatted OS IDs

252.
253.             scenar = all_scenarios[seednum]
254.             ScenarioRL=S_RL.filter(items=scenar[0:7], axis=0)
255.             ScenarioCL=S_CL.filter(items=scenar[7:13], axis=0)
256.
257.
258.             RLInds=ScenarioRL.index.tolist()
259.             CLInds=ScenarioCL.index.tolist()
260.             SD=[]
261.
262.             #Use dictonary to get scenario names
263.             for c in range(np.shape(ScenarioRL["ReservoirLevelId"])[0]):
264.                 RLID=RLev[ScenarioRL.loc[RLInds[c]]["ReservoirLevelId"]]
265.                 IMP=Impacts[ScenarioRL.loc[RLInds[c]]["ImpactTypeId"]]
266.                 OS=OSTypes[ScenarioRL.loc[RLInds[c]]["OperatingStateTypeId"]]
267.                 CF=CFType[ScenarioRL.loc[RLInds[c]]["CausalFactorTypeId"]]
268.                 SD.append(str(RLID + "-" + OS + IMP + "-" + CF + "_"))
269.
270.             for c in range(np.shape(ScenarioCL["ReservoirLevelId"])[0]):
271.                 RLID=RLev[ScenarioCL.loc[CLInds[c]]["ReservoirLevelId"]]
272.                 CLID=CLType[ScenarioCL.loc[CLInds[c]]["ComponentLevelTypeId"]]
273.                 IMP=Impacts[ScenarioCL.loc[CLInds[c]]["ImpactTypeId"]]
274.                 OS=OSTypes[ScenarioCL.loc[CLInds[c]]["OperatingStateTypeId"]]
275.                 CF=CFType[ScenarioCL.loc[CLInds[c]]["CausalFactorTypeId"]]
276.                 SD.append(str(RLID +"-"+ CLID + "-" + OS + IMP + "-
    " + CF + "_"))
277.
278.             #omitting normal conditions
279.             out=[i for i in SD if not ('-nn' in i)]
280.
281.             #Formatting as one long string explaining what's happening in scenar
    io
282.             ScenarioDescriptor="".join(out)
283.
284.             #SET RANDOM START DATES FOR SIMULATION
285.             #Between May 1 - Sept 30 thunderstorm season
286.             possible_starts = [ (x, y) for x in range( 0,364 ) for y in range( 0
    , 9999 ) ]
287.             if boolop == 1:
288.                 Starts = random.sample( possible_starts, NYr )
289.             if boolop == 0:
```

```
290.                Starts = possible_starts[ 0 : NYr ] #just takes the first bunch
    if non-randomized
291.
292.            Startdays=np.zeros(NYr)
293.            Startyears=np.zeros(NYr)
294.            for i in range(NYr):
295.                Startdays[i]=Starts[i][0]
296.                Startyears[i]=Starts[i][1]
297.
298.            refdays=dayrefs(Startdays, 0, NYr) #returns 365xNYr array containing
     reference dates from 1-365
299.
300.            #Determine unique causal factors for timing
301.            cfs=[]
302.            ScenarioRL2=ScenarioRL[ScenarioRL["ReservoirLevelId"]!=29]
303.            ScenarioRL3=ScenarioRL2[ScenarioRL2["ReservoirLevelId"]!=48]
304.            cfs.append(ScenarioRL3["CausalFactorTypeId"])
305.            #remove site access and staffing delays from cf list since they are
    implemented differently
306.            cfs.append(ScenarioCL["CausalFactorTypeId"])
307.
308.            CFS=np.zeros(int(np.size(cfs[0]))+ int(np.size(cfs[1])))
309.            for i in range(len(cfs[0])):
310.                CFS[i]=cfs[0][i]
311.            for i in range(len(cfs[1])):
312.                CFS[i+np.size(cfs[0])]=cfs[1][i]
313.            uniqueCFs=np.unique(CFS)
314.            uniqueCFs=np.setdiff1d(uniqueCFs,[25])
315.
316.            impacttimes=np.zeros((np.size(uniqueCFs), 2,NYr)) #CF, Impact Time,
    Year
317.            for i in range(NYr):
318.                if boolop==1:
319.                    random.shuffle(uniqueCFs) #does not shuffle if boolop=0 (non
     randomized)
320.                impacttimes[:, 0,i]=uniqueCFs   #This is the order of occurrence
     of the causal factor impacts for the simulation
321.                for j in range(np.size(uniqueCFs)):
322.                    impacttimes[j, 1, i]=randintswitchCF(CFDates[uniqueCFs[j]][0
    ],CFDates[uniqueCFs[j]][1], refdays[:,i], 1, deltatmax, boolop=boolop) #Sets ran
    dom times for subsequent disturbances within 6 months of original
323.                if np.size(uniqueCFs)>0:
324.                    impacttimes[0,1,i]=1 #set all first disturbances to day 1
325.            #now make a dataframe with start times for each CF
326.            imptimessorted=np.zeros((np.size(uniqueCFs), 2,NYr))
327.            for i in range(NYr):
328.                imptime=impacttimes[:,:,i]
329.                imptimessorted[:,:,i]=imptime[imptime[:,0].argsort()]
330.
331.
332.                    #Debris on gate opening. Set reduced gate capacity
333.            ReducedCapacities=1+np.zeros(( NYr)) #default no blockage
334.                    #one for each gate and inflow (can randomize degree of block
    age b/w min and max)
335.            AllDebris=ScenarioCL[ScenarioCL["ImpactTypeId"]==5]
336.            AllDebris=AllDebris.reset_index(drop=True)
337.            for c in range(np.shape(AllDebris["ReservoirLevelId"])[0]):
338.                        cavg=1.-(AllDebris["Avg"][c]/100.)
339.                        cmin=1.-(AllDebris["Min"][c]/100.)
340.                        cmax=1.-(AllDebris["Max"][c]/100.)
```

```
341.                         ReducedCapacities=gen_avg(cavg, NYr, cmin, cmax, boolop)
342.
343.             Day1Used_MtnAge=np.zeros((2,NYr))
344.
345.             #Organize outages by component
346.             GateOutagesAll=np.zeros((3, NYr)) #FC, FO, FIP
347.             GateCollapses=np.zeros((1,NYr))
348.             Gdeltat=np.zeros((3, NYr))
349.             SPOG1OutagesCL=ScenarioCL[(ScenarioCL["ImpactTypeId"]==1) & (Scenari
    oCL["ReservoirLevelId"]==13)]
350.             SPOG1OutagesCL=SPOG1OutagesCL.reset_index()
351.             for c in range (np.shape(SPOG1OutagesCL["ReservoirLevelId"])[0]):
352.
353.                 if SPOG1OutagesCL["ComponentLevelTypeId"][c]==1: #Fail closed
354.                     GateOutagesAll[0,:]=randswitch(SPOG1OutagesCL["Min"][c], SPO
    G1OutagesCL["Avg"][c], SPOG1OutagesCL["Max"][c], NYr, boolop)
355.
356.
357.
358.                 if SPOG1OutagesCL["ComponentLevelTypeId"][c]==11: #Fail open/col
    lapse
359.                     GateOutagesAll[1,:]=randswitch(SPOG1OutagesCL["Min"][c], SPO
    G1OutagesCL["Avg"][c], SPOG1OutagesCL["Max"][c], NYr, boolop)
360.                     if SPOG1OutagesCL["OperatingStateTypeId"][c]==5:
361.                         GateCollapses[0,:]=1+np.zeros(NYr)
362.
363.
364.                 if SPOG1OutagesCL["ComponentLevelTypeId"][c]==5: #Fail in place

365.                     GateOutagesAll[2,:]=randswitch(SPOG1OutagesCL["Min"][c], SPO
    G1OutagesCL["Avg"][c], SPOG1OutagesCL["Max"][c], NYr, boolop)
366.
367.
368.
369.             TurbineOutagesAll=np.zeros((2, NYr)) #HC, GEN
370.             Tdeltat=np.zeros((2,NYr))
371.             TB1OutagesCL=(ScenarioCL[ScenarioCL["ReservoirLevelId"]==8]) #filter
     to turbine only
372.             TB1OutagesCL=(TB1OutagesCL[TB1OutagesCL["OperatingStateTypeId"]!=2])
     #omit normal conditions
373.             TB1OutagesCL=TB1OutagesCL.reset_index(drop=True)
374.             for c in range (np.shape(TB1OutagesCL["ReservoirLevelId"])[0]):
375.                 if TB1OutagesCL["ComponentLevelTypeId"][c]==13: #Head Cover HC R
    ESULTS IN UNCONTROLLED RELEASE
376.                     TurbineOutagesAll[0,:]=randswitch(TB1OutagesCL["Min"][c],TB1
    OutagesCL["Avg"][c], TB1OutagesCL["Max"][c], NYr, boolop)
377.
378.                 if TB1OutagesCL["ComponentLevelTypeId"][c]==15: #Generator
379.                     TurbineOutagesAll[1,:]=randswitch(TB1OutagesCL["Min"][c],TB1
    OutagesCL["Avg"][c], TB1OutagesCL["Max"][c], NYr, boolop)
380.                     ind=int(np.where(imptimessorted[:,0,1]==TB1OutagesCL["Causal
    FactorTypeId"][c])[0])
381.
382.
383.             AllOutagesRL=(ScenarioRL[ScenarioRL["ImpactTypeId"].isin([1,9])])
384.             AllOutagesRL=AllOutagesRL.reset_index(drop=True)
385.             ScenarioRL=ScenarioRL.reset_index(drop=True)
386.             AllErrors=ScenarioRL[ScenarioRL["ImpactTypeId"]==3] #Errors to senso
    rs and Inflow Forecast
387.             AllErrors=AllErrors.reset_index(drop=True)
```

```
388.
389.            OCOutages=np.zeros((3,NYr))
390.            #PLCRTU,  Penstock, Grid
391.            SOutages=np.zeros(NYr)
392.            #Res El Sensor 1
393.
394.            OCdeltat=np.zeros((3,NYr))
395.            Sdeltat=np.zeros(NYr)
396.            for c in range (np.shape(AllOutagesRL["ReservoirLevelId"])[0]):
397.            #Other components
398.                if AllOutagesRL["ReservoirLevelId"][c]==18: #Dam PLC failure HOU
    RS
399.                    OCOutages[0, :]=randswitch(AllOutagesRL["Min"][c], AllOutage
    sRL["Avg"][c], AllOutagesRL["Max"][c], NYr, boolop)
400.
401.
402.                if AllOutagesRL["ReservoirLevelId"][c]==42: #Penstock
403.                    OCOutages[1,:]=randswitch(AllOutagesRL["Min"][c], AllOutages
    RL["Avg"][c], AllOutagesRL["Max"][c], NYr, boolop)
404.
405.                if AllOutagesRL["ReservoirLevelId"][c]==44: #CMS GRID
406.                    OCOutages[2,:]=randswitch(AllOutagesRL["Min"][c], AllOutages
    RL["Avg"][c], AllOutagesRL["Max"][c], NYr, boolop)
407.
408.            #Sensors
409.                if AllOutagesRL["ReservoirLevelId"][c]==30: #RE Sensor 1
410.                    SOutages=randswitch(AllOutagesRL["Min"][c], AllOutagesRL["Av
    g"][c], AllOutagesRL["Max"][c], NYr, boolop)
411.
412.
413.
414.            SErrors=np.zeros(NYr)
415.            SErrorDeltat=np.zeros(NYr)
416.            SErrorDuration=randintswitch2(1, 10, NYr, boolop) #randomly setting
    error duration between 1 and 10 days
417.            IFErrorDuration=randintswitch2(1, 10, NYr, boolop)
418.            IFErrorDeltat=np.zeros(NYr) #randomly setting error duration between
     1 and 10 days
419.
420.            for c in range (np.shape(AllErrors["ReservoirLevelId"])[0]):
421.            #Other components
422.                if AllErrors["ReservoirLevelId"][c]==30: #Res El Sensor 1
423.                    SErrors=randswitch(AllErrors["Min"][c], AllErrors["Avg"][c],
     AllErrors["Max"][c], NYr, boolop)
424.
425.                if AllErrors["ReservoirLevelId"][c]==45: #Inflow forecast error

426.                    ind=int(np.where(imptimessorted[:,0,1]==AllErrors["CausalFac
    torTypeId"][c])[0])
427.                    IFErrorDeltat=imptimessorted[ind,1,:]
428.
429.
430.
431.            DelayAccess=np.zeros((2, NYr))
432.            DelayStaff=np.zeros(NYr)
433.            AllDelays=ScenarioRL[ScenarioRL["ImpactTypeId"]==2]
434.            AllDelays=AllDelays.reset_index(drop=True)
435.            for c in range (np.shape(AllDelays["ReservoirLevelId"])[0]):
436.            #Other components
437.                if AllDelays["ReservoirLevelId"][c]==29: #Dam Access
```

```
438.                     DelayAccess[0, :]=randswitch(AllDelays["Min"][c], AllDelays[
    "Avg"][c], AllDelays["Max"][c], NYr, boolop)
439.
440.                 if AllDelays["ReservoirLevelId"][c]==28: #Powerhouse Access
441.                     DelayAccess[1, :]=randswitch(AllDelays["Min"][c], AllDelays[
    "Avg"][c], AllDelays["Max"][c], NYr, boolop)
442.
443.                 if AllDelays["ReservoirLevelId"][c]==48: #Powerhouse Access
444.                     DelayStaff[:]=randswitch(AllDelays["Min"][c], AllDelays["Avg
    "][c], AllDelays["Max"][c], NYr, boolop)
445.
446.
447.             """
448.             NOW DETERMINE MAX IMPACT INITIATION TIME BASED ON TIME TO REPAIR FOR
     COMPONENTS THAT CAUSE A LOSS IN CAPACITY
449.             """
450.             #figure out length of time components are out for...
451.             deltatmax=GateOutagesAll[0,:] + GateOutagesAll[1,:] + GateOutagesAll
    [2,:] + TurbineOutagesAll[1,:] + TurbineOutagesAll[0,:] + OCOutages[0,:] + OCOut
    ages[1,:]+ OCOutages[2,:] + SErrorDuration + SErrors
452.             deltatmax=deltatmax*0.8  #somewhat arbitrary. Can experiment and sel
    ect to ensure enough data points are collected for each scenario
453.             deltatmax=deltatmax.clip(4, 180)
454.
455.
456.             """
457.             NOW THAT WE HAVE SET DELTATMAX, WE CAN SET THE IMPACT TIMES
458.             """
459.
460.             for c in range (np.shape(SPOG1OutagesCL["ReservoirLevelId"])[0]):
461.
462.                 if SPOG1OutagesCL["ComponentLevelTypeId"][c]==1: #Fail closed
463.                     ind=int(np.where(imptimessorted[:,0,1]==SPOG1OutagesCL["Caus
    alFactorTypeId"][c])[0])
464.                     Gdeltat[0,:]=imptimessorted[ind,1,:]
465.                     if imptimessorted[ind,0,0]==2:  #mtnce
466.                         for y in range(NYr):
467.                             if imptimessorted[ind,1,y]>1:
468.                                 Gdeltat[0,y]=randintswitch(0, deltatmax, 1, bool
    op)
469.                             else:
470.                                 if Day1Used_MtnAge[0,y]==1:
471.                                     Gdeltat[0,y]=randintswitch(0, deltatmax, 1,
    boolop) #set all mtnce events randomly except the first occurrence of a day 1 mt
    nce failure
472.                                 else:
473.                                     Gdeltat[0,y]=1
474.                                     Day1Used_MtnAge[0,y]=1
475.
476.                     if imptimessorted[ind,0,0]==9:  #aging
477.                         for y in range(NYr):
478.                             if imptimessorted[ind,1,y]>1:
479.                                 Gdeltat[0,y]=randintswitch(0, deltatmax, 1, bool
    op) #set all aging events randomly except the first occurrence of a day 1 age fa
    ilure
480.                             else:
481.                                 if Day1Used_MtnAge[1,y]==1:
482.                                     Gdeltat[0,y]=randintswitch(0, deltatmax, 1,
    boolop)
483.                                 else:
484.                                     Gdeltat[0,y]=1
```

```
                                    Day1Used_MtnAge[1,y]=1


                if SPOG1OutagesCL["ComponentLevelTypeId"][c]==11: #Fail open/col
    lapse
                    ind=int(np.where(imptimessorted[:,0,1]==SPOG1OutagesCL["Caus
    alFactorTypeId"][c])[0])
                    Gdeltat[1,:]=imptimessorted[ind,1,:]
                    if SPOG1OutagesCL["OperatingStateTypeId"][c]==5:
                        GateCollapses[0,:]=1+np.zeros(NYr)
                    if imptimessorted[ind,0,0]==2:  #mtnce
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                Gdeltat[1,y]=randintswitch(0, deltatmax, 1, bool
    op)
                            else:
                                if Day1Used_MtnAge[0,y]==1:
                                    Gdeltat[1,y]=randintswitch(0, deltatmax, 1,
    boolop) #set all mtnce events randomly except the first occurrence of a day 1 mt
    nce failure
                                else:
                                    Gdeltat[1,y]=1
                                    Day1Used_MtnAge[0,y]=1
                    if imptimessorted[ind,0,0]==9:  #aging
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                Gdeltat[1,y]=randintswitch(0, deltatmax, 1, bool
    op) #set all aging events randomly except the first occurrence of a day 1 age fa
    ilure
                            else:
                                if Day1Used_MtnAge[1,y]==1:
                                    Gdeltat[1,y]=randintswitch(0, deltatmax, 1,
    boolop)
                                else:
                                    Gdeltat[1,y]=1
                                    Day1Used_MtnAge[1,y]=1


                if SPOG1OutagesCL["ComponentLevelTypeId"][c]==5: #Fail in place

                    ind=int(np.where(imptimessorted[:,0,1]==SPOG1OutagesCL["Caus
    alFactorTypeId"][c])[0])
                    Gdeltat[2,:]=imptimessorted[ind,1,:]
                    if imptimessorted[ind,0,0]==2:  #mtnce
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                Gdeltat[2,y]=randintswitch(0, deltatmax, 1, bool
    op)
                            else:
                                if Day1Used_MtnAge[0,y]==1:
                                    Gdeltat[2,y]=randintswitch(0, deltatmax, 1,
    boolop) #set all mtnce events randomly except the first occurrence of a day 1 mt
    nce failure
                                else:
                                    Gdeltat[2,y]=1
                                    Day1Used_MtnAge[0,y]=1
                    if imptimessorted[ind,0,0]==9:  #aging
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
```

```
                                Gdeltat[2,y]=randintswitch(0, deltatmax, 1, bool
    op) #set all aging events randomly except the first occurrence of a day 1 age fa
    ilure
                            else:
                                if Day1Used_MtnAge[1,y]==1:
                                    Gdeltat[2,y]=randintswitch(0, deltatmax, 1,
    boolop)
                                else:
                                    Gdeltat[2,y]=1
                                    Day1Used_MtnAge[1,y]=1


            for c in range (np.shape(TB1OutagesCL["ReservoirLevelId"])[0]):
                if TB1OutagesCL["ComponentLevelTypeId"][c]==13: #Head Cover HC R
    ESULTS IN UNCONTROLLED RELEASE
                    ind=int(np.where(imptimessorted[:,0,1]==TB1OutagesCL["Causal
    FactorTypeId"][c])[0])
                    Tdeltat[0,:]=imptimessorted[ind,1,:]
                    if imptimessorted[ind,0,0]==2:  #mtnce
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                Tdeltat[0,y]=randintswitch(0, deltatmax, 1, bool
    op)
                            else:
                                if Day1Used_MtnAge[0,y]==1:
                                    Tdeltat[0,y]=randintswitch(0, deltatmax, 1,
    boolop) #set all mtnce events randomly except the first occurrence of a day 1 mt
    nce failure
                                else:
                                    Tdeltat[0,y]=1
                                    Day1Used_MtnAge[0,y]=1
                    if imptimessorted[ind,0,0]==9:  #aging
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                Tdeltat[0,y]=randintswitch(0, deltatmax, 1, bool
    op) #set all aging events randomly except the first occurrence of a day 1 age fa
    ilure
                            else:
                                if Day1Used_MtnAge[1,y]==1:
                                    Tdeltat[0,y]=randintswitch(0, deltatmax, 1,
    boolop)
                                else:
                                    Tdeltat[0,y]=1
                                    Day1Used_MtnAge[1,y]=1

                if TB1OutagesCL["ComponentLevelTypeId"][c]==15: #Generator
                    ind=int(np.where(imptimessorted[:,0,1]==TB1OutagesCL["Causal
    FactorTypeId"][c])[0])
                    Tdeltat[1,:]=imptimessorted[ind,1,:]
                    if imptimessorted[ind,0,0]==2:  #mtnce
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                Tdeltat[1,y]=randintswitch(0, deltatmax, 1, bool
    op)
                            else:
                                if Day1Used_MtnAge[0,y]==1:
                                    Tdeltat[1,y]=randintswitch(0, deltatmax, 1,
    boolop) #set all mtnce events randomly except the first occurrence of a day 1 mt
    nce failure
                                else:
```

```
                                    Tdeltat[1,y]=1
                                    Day1Used_MtnAge[0,y]=1
                    if imptimessorted[ind,0,0]==9:  #aging
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                Tdeltat[1,y]=randintswitch(0, deltatmax, 1, bool
op) #set all aging events randomly except the first occurrence of a day 1 age fa
ilure
                            else:
                                if Day1Used_MtnAge[1,y]==1:
                                    Tdeltat[1,y]=randintswitch(0, deltatmax, 1,
boolop)
                                else:
                                    Tdeltat[1,y]=1
                                    Day1Used_MtnAge[1,y]=1


            for c in range (np.shape(AllOutagesRL["ReservoirLevelId"])[0]):
            #Other components
                if AllOutagesRL["ReservoirLevelId"][c]==18: #Dam PLC failure HOU
RS
                    ind=int(np.where(imptimessorted[:,0,1]==AllOutagesRL["Causal
FactorTypeId"][c])[0])
                    OCdeltat[0,:]=imptimessorted[ind,1,:]
                    if imptimessorted[ind,0,0]==2:  #mtnce
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                OCdeltat[0,y]=randintswitch(0, deltatmax, 1, boo
lop)
                            else:
                                if Day1Used_MtnAge[0,y]==1:
                                    OCdeltat[0,y]=randintswitch(0, deltatmax, 1,
 boolop) #set all mtnce events randomly except the first occurrence of a day 1 m
tnce failure
                                else:
                                    OCdeltat[0,y]=1
                                    Day1Used_MtnAge[0,y]=1
                    if imptimessorted[ind,0,0]==9:  #aging
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                OCdeltat[0,y]=randintswitch(0, deltatmax, 1, boo
lop) #set all aging events randomly except the first occurrence of a day 1 age f
ailure
                            else:
                                if Day1Used_MtnAge[1,y]==1:
                                    OCdeltat[0,y]=randintswitch(0, deltatmax, 1,
 boolop)
                                else:
                                    OCdeltat[0,y]=1
                                    Day1Used_MtnAge[1,y]=1


                if AllOutagesRL["ReservoirLevelId"][c]==42: #Penstock
                    ind=int(np.where(imptimessorted[:,0,1]==AllOutagesRL["Causal
FactorTypeId"][c])[0])
                    OCdeltat[1,:]=imptimessorted[ind,1,:]
                    if imptimessorted[ind,0,0]==2:  #mtnce
                        for y in range(NYr):
                            if imptimessorted[ind,1,y]>1:
                                OCdeltat[1,y]=randintswitch(0, deltatmax, 1, boo
lop)
```

```
                            else:
                                if Day1Used_MtnAge[0,y]==1:
                                    OCdeltat[1,y]=randintswitch(0, deltatmax, 1,
     boolop) #set all mtnce events randomly except the first occurrence of a day 1 m
    tnce failure
                                else:
                                    OCdeltat[1,y]=1
                                    Day1Used_MtnAge[0,y]=1
                        if imptimessorted[ind,0,0]==9:  #aging
                            for y in range(NYr):
                                if imptimessorted[ind,1,y]>1:
                                    OCdeltat[1,y]=randintswitch(0, deltatmax, 1, boo
    lop) #set all aging events randomly except the first occurrence of a day 1 age f
    ailure
                            else:
                                if Day1Used_MtnAge[1,y]==1:
                                    OCdeltat[1,y]=randintswitch(0, deltatmax, 1,
     boolop)
                                else:
                                    OCdeltat[1,y]=1
                                    Day1Used_MtnAge[1,y]=1

                    if AllOutagesRL["ReservoirLevelId"][c]==44: #CMS GRID
                        ind=int(np.where(imptimessorted[:,0,1]==AllOutagesRL["Causal
    FactorTypeId"][c])[0])
                        OCdeltat[2,:]=imptimessorted[ind,1,:]
                        if imptimessorted[ind,0,0]==2:  #mtnce
                            for y in range(NYr):
                                if imptimessorted[ind,1,y]>1:
                                    OCdeltat[2,y]=randintswitch(0, deltatmax, 1, boo
    lop)
                            else:
                                if Day1Used_MtnAge[0,y]==1:
                                    OCdeltat[2,y]=randintswitch(0, deltatmax, 1,
     boolop) #set all mtnce events randomly except the first occurrence of a day 1 m
    tnce failure
                                else:
                                    OCdeltat[2,y]=1
                                    Day1Used_MtnAge[0,y]=1
                        if imptimessorted[ind,0,0]==9:  #aging
                            for y in range(NYr):
                                if imptimessorted[ind,1,y]>1:
                                    OCdeltat[2,y]=randintswitch(0, deltatmax, 1, boo
    lop) #set all aging events randomly except the first occurrence of a day 1 age f
    ailure
                            else:
                                if Day1Used_MtnAge[1,y]==1:
                                    OCdeltat[2,y]=randintswitch(0, deltatmax, 1,
     boolop)
                                else:
                                    OCdeltat[2,y]=1
                                    Day1Used_MtnAge[1,y]=1

                #Sensors
                    if AllOutagesRL["ReservoirLevelId"][c]==30: #RE Sensor 1
                        ind=int(np.where(imptimessorted[:,0,1]==AllOutagesRL["Causal
    FactorTypeId"][c])[0])
                        Sdeltat=imptimessorted[ind,1,:]
                        if imptimessorted[ind,0,0]==2:  #mtnce
                            for y in range(NYr):
                                if imptimessorted[ind,1,y]>1:
```

```
                                    Sdeltat[y]=randintswitch(0, deltatmax, 1, boolop
    )
                                else:
                                    if Day1Used_MtnAge[0,y]==1:
                                        Sdeltat[y]=randintswitch(0, deltatmax, 1, bo
    olop) #set all mtnce events randomly except the first occurrence of a day 1 mtnc
    e failure
                                    else:
                                        Sdeltat[y]=1
                                        Day1Used_MtnAge[0,y]=1
                            if imptimessorted[ind,0,0]==9:  #aging
                                for y in range(NYr):
                                    if imptimessorted[ind,1,y]>1:
                                        Sdeltat[y]=randintswitch(0, deltatmax, 1, boolop
    ) #set all aging events randomly except the first occurrence of a day 1 age fail
    ure
                                else:
                                    if Day1Used_MtnAge[1,y]==1:
                                        Sdeltat[y]=randintswitch(0, deltatmax, 1, bo
    olop)
                                    else:
                                        Sdeltat[y]=1
                                        Day1Used_MtnAge[1 ,y]=1



                    for c in range (np.shape(AllErrors["ReservoirLevelId"])[0]):
                    #Other components
                        if AllErrors["ReservoirLevelId"][c]==30: #Res El Sensor 1
                            ind=int(np.where(imptimessorted[:,0,1]==AllErrors["CausalFac
    torTypeId"][c])[0])
                            SErrorDeltat=imptimessorted[ind,1,:]

                        if AllErrors["ReservoirLevelId"][c]==45: #Inflow forecast error

                            ind=int(np.where(imptimessorted[:,0,1]==AllErrors["CausalFac
    torTypeId"][c])[0])
                            IFErrorDeltat=imptimessorted[ind,1,:]



        """
        3. INITIALIZING SUPPORTING FUNCTIONS AND ARRAYS

        -Sets up arrays to be populated by SD model
        -Sets up supporting functions
        -
    Functions defined here are not part of the System Dynamics model but may be call
    ed from it

        """

        runname="Simple64-i1-O-2018-5yr"
        runname1=runname

        start=(str(datetime.now()))

        @njit
```

```
724.        def SSC(storage): #Stage storage curve
725.            if storage<1400:
726.                return -1.1201e-
    05 * storage**2 + 0.032473 * storage + 364.6572
727.            else:
728.                return 388.16
729.
730.        @njit
731.        def SSCrev(stage): #Storage stage curve
732.            return np.roots(np.array([-1.1201e-05, 0.032473, 364.6572-
    stage]))[1]
733.        @njit
734.        def OTC(elev): #Overtopping curve
735.            if elev<=380.41: #378.41
736.                return 0
737.            else:
738.                return (-35.7505780379803*(elev-2)**3 + 40896.2749435669*(elev-
    2)**2 -15593240.0619064*(elev-2) + 1981715583.08889)*0.3
739.
740.
741.        #Rating curves for different gates to be used to switch between gate pos
    ition and flow
742.        RatingCurve1=pd.read_csv(datafile_path("SPOGAllRC.csv"), index_col=0)
743.        x1=np.asarray(RatingCurve1.index.values, dtype=float)
744.        y1=np.asarray(RatingCurve1.columns.values, dtype=float)
745.        z1=np.asarray(RatingCurve1.values, dtype=float)
746.
747.        @njit
748.        def fncTurbineMaxFlow(elev, flagT):
749.            if flagT==1: #// Turb on
750.                if elev < 363.05:
751.                    result = 0
752.                elif 363.05 <= elev < 365.05:
753.                    result = 13.98 * elev - 5075.44
754.                elif 365.05 <= elev < 367.05:
755.                    result = 18.02334 * elev - 6551.46
756.                else:
757.                    result = 65
758.            else: #//both off
759.                return 0
760.            return result
761.
762.        @njit
763.        def SPOGMaxFlow(elev, flag): #Sums the values from two gates into a sing
    le gate discharge
764.            if flag == 1:
765.                if 367.28<=elev<367.5:
766.                    out1= 19.09091*elev-7011.7
767.                if 367.5<=elev<=367.8:
768.                    out1= 37.33334*elev-13715.8
769.                if 367.8<=elev<=369:#368.1:
770.                    out1= 49.667*elev-18252
771.                if 369 <=elev < 381.6:  #367.8 sill
772.                    out1= 2.154624239*elev**2 - 1496.3410084*elev + 258875.37647
    999998
773.                elif elev >= 381.6:
774.                    out1= 861.1+728.9
775.                else:
776.                    out1= 0
777.            else:
778.                out1=0
```

```
            return out1

        @njit
        def GateInstr(ResEl, OP):
            y=y1
            z=z1
            x=x1
            GatePosition=0
            if (ResEl > 367.28):
                Yo=np.abs(y-ResEl).argsort()[0:2]
                WtYo0=np.abs((ResEl-y[Yo[0]])/(y[Yo[0]]-y[Yo[1]]))
                WtYo1=np.abs((ResEl-y[Yo[1]])/(y[Yo[0]]-y[Yo[1]]))
                ResElFlow=(1-WtYo0)*z[:,Yo[0]]+(1-WtYo1)*z[:,Yo[1]]
                GateFlow=np.round(OP,2)
                GateFlowMax=np.max(ResElFlow)
                if GateFlow<=0:
                    return 0
                else:
                    if (GateFlow>GateFlowMax):
                        GateFlow=GateFlowMax-0.01
                    if GateFlow < ResElFlow[0]:
                        return x[0]
                    elif GateFlow > ResElFlow[-1]:
                        return x[-1]
                    else:
                        for i in range(len(ResElFlow) - 1):
                            if ResElFlow[i] <= GateFlow <= ResElFlow[i + 1]:
                                X1, X2 = ResElFlow[i], ResElFlow[i + 1]
                                Y1, Y2 = x[i], x[i + 1]

                                return Y1 + (Y2 - Y1) / (X2 - X1) * (GateFlow -
    X1)
            else:
                return GatePosition

        @njit
        def interpolate(x0, x, y):
            if x0 < x[0]:
                return y[0]
            elif x0 > x[-1]:
                return y[-1]
            else:
                for i in range(len(x) - 1):
                    if x[i] <= x0 <= x[i + 1]:
                        x1, x2 = x[i], x[i + 1]
                        y1, y2 = y[i], y[i + 1]

                        return y1 + (y2 - y1) / (x2 - x1) * (x0 - x1)

        @njit
        def GateFlowCalc(ResEl, GP):
            y=y1
            z=z1
            x=x1
            GateFlow=0
            if (ResEl > 367.28):
                if GP>12.5:
                    GP=12.4999
                Yo=np.abs(y-ResEl).argsort()[0:2]
                WtYo0=np.abs((ResEl-y[Yo[0]])/(y[Yo[0]]-y[Yo[1]]))
                WtYo1=np.abs((ResEl-y[Yo[1]])/(y[Yo[0]]-y[Yo[1]]))
```

```
839.                ResElFlow=(1-WtYo0)*z[:,Yo[0]]+(1-WtYo1)*z[:,Yo[1]]
840.        #        GateFlow=interpolate(GP, x, ResElFlow)
841.                if GP < x[0]:
842.                    return ResElFlow[0]
843.                elif GP > x[-1]:
844.                    return ResElFlow[-1]
845.                else:
846.                    for i in range(len(x) - 1):
847.                        if x[i] <= GP <= x[i + 1]:
848.                            X1, X2 = x[i], x[i + 1]
849.                            Y1, Y2 = ResElFlow[i], ResElFlow[i + 1]
850.
851.                            return Y1 + (Y2 - Y1) / (X2 - X1) * (GP - X1)
852.            else:
853.                return GateFlow
854.
855.        #Storage min and max
856.        Smin=8.864837907352
857.        Smax=516.35
858.
859.        #Set arrays to save model outputs for NYr years of inflows
860.        year=0
861.        RSEs=np.zeros((365,NYr))
862.        TBFs=np.zeros((365,NYr))
863.        SPOGs=np.zeros((365,NYr))
864.        OT=np.zeros((365,NYr))
865.        INFs=np.zeros((365,NYr))
866.        OUTFs=np.zeros((365,NYr))
867.        GPs=np.zeros((365, NYr))
868.        GAVs=np.zeros((365, NYr))
869.        UAVs=np.zeros((365, NYr))
870.        MOBI=np.zeros((365, NYr))
871.        MOB=np.zeros((365, NYr))
872.        TOTR=np.zeros((365,NYr))
873.        DEBRISREMOVAL=np.zeros(NYr)
874.        DAY=np.zeros((365,NYr))
875.        MON=np.zeros((365,NYr))
876.        AllMaxQ_t=np.zeros((365,2, NYr))
877.        AllMaxQ=[861.1+728.9,32.5+32.5]
878.        TTRS=np.zeros((365,8, NYr))
879.        Retention=np.zeros((365,NYr))
880.        yearnum=np.zeros(NYr)
881.        for yr in range(NYr):
882.            yearnum[yr]=str(1984+yr)
883.        GateCaps=np.zeros((365, NYr))
884.        CAPs=np.zeros((365, NYr))
885.        EOCs=np.zeros((365, NYr))
886.        UCRs=np.zeros((365,NYr))
887.        GCRs=np.zeros((365,NYr))
888.        GAVs=np.zeros((365, NYr))
889.        OSDs=-1+np.zeros((36,NYr)) #36 component outage start dates
890.        OLs=np.zeros((36,NYr))
891.        #Inflow forecast accuracy data
892.        #ForecastError=pd.read_csv(datafile_path("CMSForecastError.csv"), index_
    col=0)
893.        #day=ForecastError.index.values #day of forecast
894.        #errordata=ForecastError.values #error mean, mean over 110cms and standa
    rd deviation, std over 110cms
895.        #MAEt=interp1d(day,errordata[:,0])
896.        #MAE110t=interp1d(day,errordata[:,1])
897.        #SEt=interp1d(day,errordata[:,2])
```

```
898.        #SE110t=interp1d(day,errordata[:,3])
899.        Fish=pd.read_csv(datafile_path("Fish.csv"), header=0).values[:,1].astype
    ("float64")
900.
901.        @njit
902.        def getfishflow(dayref):
903.            return Fish[int(dayref[0]):int(dayref[0]+14)]
904.        #Fish=np.zeros((3,3))
905.        #Fish-[0,:]=[5,7,3]
906.        #Fish [1,:]=[0,90,304]
907.        #Fish[2,:]=[89,303,365]
908.        #
909.        #@njit
910.        #def getfishflow(dayref):
911.        #    return Fish[int(dayref[0]):int(dayref[0]+14)]
912.        ##    ff=np.zeros(14)
913.        ##    for i in range(14):
914.        ##        if dayref[i]<90:
915.        ##            ff[i]=5
916.        ##        elif dayref[i]<304:
917.        ##            ff[i]=7
918.        ##        elif dayref[i]<365:
919.        ##            ff[i]=3
920.        ##    return ff
921.
922.        @njit
923.        def dayrefs(Startdays, timestep):
924.            daynum=Startdays+int(timestep) #STARTDAYS CONTAINS 1-
    365 STARTING DAY REF
925.            if daynum>365:
926.                daynum=daynum-365
927.            dayref=np.zeros(14)
928.            dayref[0]=daynum
929.            for t in range(13):  #Converts vensim date into numbers 1-
    365 to represent dates in the model
930.                if dayref[t]+1<366:
931.                    dayref[t+1]=dayref[t]+1
932.                else: dayref[t+1]=1
933.            return dayref
934.
935.
936.        #SETTING UP RANDOM SIMULATION START POINTS AND ASSIGNING BASELINE CONDIT
    IONS FROM "NORMAL" OPS
937.
938.
939.        if seedgen==0:
940.            Startdays=Seeds['Startdays']
941.            Starts=Seeds['Starts'] #np.transpose(Seeds["Starts"]) for jan 1 star
    ts
942.
943.
944.        #Starts=np.transpose(Starts)
945.        daynum=Startdays.copy()
946.        OutputDayrefs=np.zeros((365,NYr))
947.        OutputDayrefs[0, :]=daynum
948.        for t in range(364):  #Converts vensim date into numbers 1-
    365 to represent dates in the model
949.            for yr in range(NYr):
950.                if OutputDayrefs[t, yr]+1<366:
951.                    OutputDayrefs[t+1, yr]=OutputDayrefs[t, yr]+1
952.                else: OutputDayrefs[t+1, yr]=1
```

```
953.
954.        StartRSEs=np.zeros(NYr)
955.        for i in range(NYr):
956.            StartRSEs[i]=NormalRSEs[Starts[i][0], Starts[i][1]] #for jan 1 start
    s
957.            #StartRSEs[i]=NormalRSEs[Starts[0,i], Starts[1,i]] #for SeedsS_Nov06
    _2018
958.
959.        #SET UP INFLOWS AND BASELINE
960.
961.        Inflow=np.zeros((730,NYr)) #2 year min
962.        B_RSEs = np.zeros((365, NYr))
963.        B_TBFs = np.zeros((365, NYr))
964.        B_TSs = np.zeros((365, NYr))
965.        B_OTs = np.zeros((365, NYr))
966.        for i in range(NYr):
967.            startdayind=Starts[i][0]
968.            Inflow[0:int(730-
    startdayind),i]=InflowJan1Start[startdayind:730,Starts[i][1]]
969.            Inflow[int(730-
    startdayind):730,i]=InflowJan1Start[0:int(startdayind),Starts[i][1]+1]
970.            B_RSEs[0:int(365-
    startdayind),i]=NormalRSEs[startdayind:365,Starts[i][1]]
971.            B_RSEs[int(365-
    startdayind):365,i]=NormalRSEs[0:int(startdayind),Starts[i][1]+1]
972.            B_TBFs[0:int(365-
    startdayind),i]=NormalTBFs[startdayind:365,Starts[i][1]+1]
973.            B_TBFs[int(365-
    startdayind):365,i]=NormalTBFs[0:int(startdayind),Starts[i][1]+2]
974.            B_TSs[0:int(365-
    startdayind),i]=NormalTSs[startdayind:365,Starts[i][1]+1]
975.            B_TSs[int(365-
    startdayind):365,i]=NormalTSs[0:int(startdayind),Starts[i][1]+2]
976.            B_OTs[0:int(365-
    startdayind),i]=NormalOTs[startdayind:365,Starts[i][1]+1]
977.            B_OTs[int(365-
    startdayind):365,i]=NormalOTs[0:int(startdayind),Starts[i][1]+2]
978.
979.        if seedgen==0:
980.            ReducedCapacities=Seeds['ReducedCapacities']
981.
982.        ReducedCapacityMinimumTime=10 #10 days to arrange debris removal, at a m
    inimum
983.        global DebrisRemoval
984.        DebrisRemoval=0
985.
986.        Inf114=np.zeros(14)
987.
988.        #This is used to ensure inflow forecast and ops planning done once per 2
    4 hours (at midnight)
989.        #def isinterger(number):
990.        #    return np.equal(np.mod(number, 1), 0)
991.
992.
993.        #@njit
994.        #def getmaxq(component, ResEl): #returns the maximum available discharge
     for a given component for all res els
995.        #    if component==0: #GATE 1
996.        #        GateFlowMax=SPOGMaxFlow(ResEl, 1)
997.        #    elif component==2: #TURBINE 1g
998.        #        GateFlowMax=fncTurbineMaxFlow(ResEl, 1)
```

```
999.        #    return GateFlowMax
1000.
1001.       @njit
1002.       def fncSPOGMaxFlow(elev, flag, El1d):
1003.           if flag == 1:
1004.               if 367.28<=elev<367.5:
1005.                   out1= 9.545455*elev-3505.85
1006.                   out2= 9.545455*elev-3505.85
1007.               if 367.5<=elev<=367.8:
1008.                   out1= 18.66667*elev-6857.9
1009.                   out2= 18.66667*elev-6857.9
1010.               if 367.8<=elev<=369:#368.1:
1011.                   out1= 25*elev-9187.3
1012.                   out2= 24.66667*elev-9064.7
1013.               if 369 <=elev < 381.6:  #367.8 sill
1014.                   out1= 1.494595567*elev**2 - 1056.252204*elev + 186302.1873
1015.                   out2= 0.660028672*elev**2 - 440.0888044*elev + 72573.18918
1016.               if elev >= 381.6:
1017.                   out1= 861.1
1018.                   out2= 728.9
1019.               if elev<367.28:
1020.                   out1= 0
1021.                   out2=0
1022.               if El1d>376.5: #corects max flow for extreme high inflow events

1023.                   elev=(elev+376.50)/2.
1024.                   out1= 1.494595567*elev**2 - 1056.252204*elev + 186302.1873
1025.                   out2= 0.660028672*elev**2 - 440.0888044*elev + 72573.18918
1026.
1027.           else:
1028.               out1=0
1029.               out2=0
1030.           return out1+out2
1031.
1032.       @njit
1033.       def availarray(length):
1034.           out=np.ones(14)
1035.           if length>0:
1036.               out[0:length]=0
1037.           return out
1038.
1039.       @njit
1040.       def OpsPlan(InflowForecast, Storage, dayref, SPG1Av,  TbAv1, resElPens):

1041.           FishFlow=getfishflow(dayref)
1042.           SPOG1Av=availarray(int(SPG1Av))
1043.           TurbAv1=availarray(int(TbAv1))
1044.
1045.           VolInflow=np.sum(InflowForecast)
1046.           #FIRST ASSUME SPILL EQUAL TO FISH FOW
1047.           Spill=min(FishFlow[0],
1048.                                    fncSPOGMaxFlow(SSC(Storage), SPOG1Av[0], 373)
    )
1049.           #Assume power flow equal to max. of difference between inflow an fis
    h flow, or total available turbine flow
1050.           PFlow=max(min(fncTurbineMaxFlow(SSC(Storage), TurbAv1[0]), InflowFor
    ecast[0]-Spill), 0)
1051.
1052.           #Now check multi-day reservoir elevation
1053.           Spl=Spill+np.zeros(14)
1054.           Pow=PFlow+np.zeros(14)
```

```
1055.           HiRes=0
1056.           LoRes=0
1057.           for i in range(13):
1058.               VolOut=(i+1)*(Spill+PFlow)
1059.               VolInflow=np.sum(InflowForecast[0:i+1])
1060.               StorageD=Storage+VolInflow-VolOut
1061.               SLimitsD=GetNMax(resElPens[0,:], resElPens[2,:], resElPens[1,:],
     dayref[i])
1062.
1063.               if StorageD>SLimitsD[1]: #If 14 day storage exceeds nmax
1064.                   HiRes+=1
1065.                   #ensure power flow is max:
1066.                   Pow[i]=min(PFlow+(StorageD-
    SLimitsD[1])*(1./(i+1)),fncTurbineMaxFlow(SSC(Storage), TurbAv1[0]))
1067.                   #recalculate and recheck
1068.                   VolOut=(i+1)*(Spl[i]+Pow[i])
1069.                   StorageD=Storage+VolInflow-VolOut
1070.                   if StorageD>SLimitsD[1]: #add 1/14 of difference each day to
     spill to bring res el down
1071.                       Spl[i]=min(Spill+(StorageD-
    SLimitsD[1])*(1./(i+1)), fncSPOGMaxFlow(SSC(Storage), SPOG1Av[0], SSC(StorageD))
    )
1072.
1073.               if StorageD<SLimitsD[0]: #If final storage less than nmin
1074.                   LoRes+=1
1075.                   #ensure spill is min
1076.                   Spl[i]=min(FishFlow[0],
1077.                         fncSPOGMaxFlow(SSC(Storage), SPOG1Av[0], SSC(StorageD)
    ))
1078.                   #recalulate and recheck
1079.                   VolOut=(i+1)*(Spl[i]+Pow[i])
1080.                   StorageD=Storage+VolInflow-VolOut
1081.                   if StorageD<SLimitsD[0]: #reduce power flow to conserve wate
    r
1082.                       Pow[i]=max(PFlow-(1./(i+1))*(SLimitsD[0]-StorageD), 0)
1083.
1084.
1085.
1086.           if HiRes>0:
1087.               Spill=np.max(Spl)
1088.               PFlow=np.max(Pow) #high reservoir levels trump low reservoir lev
    els
1089.
1090.           else:
1091.               if LoRes>0:
1092.                   Spill=np.min(Spl)
1093.                   PFlow=np.min(Pow)
1094.               else:
1095.                   Spill=Spl[0]
1096.                   PFlow=Pow[0]
1097.
1098.
1099.           #check day 1 elevs again
1100.           Storage1d=Storage+InflowForecast[0]-Spill-PFlow
1101.           SLimitsD=GetNMax(resElPens[0,:], resElPens[2,:], resElPens[1,:], day
    ref[0])
1102.           if Storage1d>SLimitsD[1]:
1103.               #increase power
1104.               PFlow=fncTurbineMaxFlow(SSC(Storage), TurbAv1[0])
1105.               #recalculate
1106.               Storage1d=Storage+InflowForecast[0]-Spill-PFlow
```

```
1107.             if Storage1d>SLimitsD[1]:
1108.
1109.             #increase spill more
1110.                 spl=Spill
1111.                 Spill=min(Spill+(Storage1d-
    SLimitsD[1]), fncSPOGMaxFlow(SSC(Storage), SPOG1Av[0], SSC(Storage1d)))
1112.                 if SPOG1Av[0]==1 and Spill<spl+(Storage1d-SLimitsD[1]):
1113.                     Spill=min(Spill+(Storage1d-SLimitsD[1]), 1590)
1114.                 #recalculate
1115.                 Storage1d=Storage+InflowForecast[0]-Spill-PFlow
1116.                 if Storage1d>SLimitsD[1]+0.1: #if inflow causes reservoir to
     rise to extreme levels within 1 ts
1117.                     Spill=min(Spill+(Storage1d-
    SLimitsD[1]), fncSPOGMaxFlow(SSC((Storage+Storage1d)/2), SPOG1Av[0], SSC(Storage
    1d)))
1118.                     Storage1d=Storage+InflowForecast[0]-Spill-PFlow
1119.
1120.         if Storage1d<SLimitsD[0]:
1121.             #decrease spill
1122.             Spill=max(Spill-(SLimitsD[0]-Storage1d), FishFlow[0])
1123.             #recalculate
1124.             Storage1d=Storage+InflowForecast[0]-Spill-PFlow
1125.             if Storage1d<SLimitsD[0]:
1126.                 PFlow=max(PFlow-(SLimitsD[0]-Storage1d), 0)
1127.
1128.         if Spill<FishFlow[0]:
1129.             if SPOG1Av[0]==1: #If spill less than FF and spillway is availab
    le, readjust SPOG flow and Pflow
1130.                 Spill2=min(FishFlow[0], fncSPOGMaxFlow(SSC(Storage), SPOG1Av
    [0], SSC(Storage1d)))
1131.                 PFlow=max(fncTurbineMaxFlow(SSC(Storage), TurbAv1[0]), PFlow
    -(Spill2-Spill))
1132.
1133.         Spill2=Spill
1134.         #Now allocate discharge
1135.         Ops=[0,0] #SPOG1,Turb1
1136.         if SPOG1Av[0]==1:
1137.             Ops[0]=min(Spill2, fncSPOGMaxFlow(SSC(Storage), SPOG1Av[0], SSC(
    Storage1d)))
1138.             if SPOG1Av[0]==1 and Spill2>Ops[0]: #this helps with large inflo
    w events, where the initial gate capacity is too low, but the reservoir ends up
    too high
1139.                 Ops[0]=Spill2
1140.             if (SSC(Storage)<367.28):
1141.                 Ops[0]=0
1142.         elif SPOG1Av[0]==0:
1143.             if SPOG1Av[0]==1:
1144.                 Ops[0]=0
1145.
1146.         if TurbAv1[0]==1:
1147.             Ops[1]=PFlow
1148.
1149.         elif TurbAv1[0]==0:
1150.             Ops[1]=0
1151.
1152.
1153.         return Ops
1154.
1155.     @njit
1156.     def GetNMax(lstResLimitDays, VResLower, VResUpper, dayref):
1157.         i = -1
```

```
1158.           ColumnsCount = VResLower.shape[0]
1159.           if (lstResLimitDays[1] > dayref >= lstResLimitDays[0]):
1160.               i = 0
1161.           if (dayref>= lstResLimitDays[1]):
1162.               i = 1
1163.           if (dayref >= lstResLimitDays[2]):
1164.               i = 2
1165.           if (ColumnsCount==4):
1166.               if (dayref >= lstResLimitDays[3]):
1167.                   i = 3
1168.           if (ColumnsCount==5):
1169.               if (dayref >= lstResLimitDays[4]):
1170.                   i = 4
1171.           if (ColumnsCount==6):
1172.               if (dayref >= lstResLimitDays[5]):
1173.                   i = 5
1174.           if (ColumnsCount==7):
1175.               if (dayref >= lstResLimitDays[6]):
1176.                   i = 6
1177.           return (VResLower[i], VResUpper[i])
1178.
1179.
1180.       #Determines month and day of year so fish flows and res el penalties cor
    respond to timing
1181.
1182.       if seedgen==0:
1183.           ScenarioDescriptor=Seeds["ScenarioDescriptor"]
1184.
1185.
1186.       """
1187.       4. DEFINING sdpy FUNCTIONS
1188.
1189.       Broken down sector-by-sector:
1190.           4.1. Hydaulic System State
1191.           4.2. Sensors
1192.           4.3. Disturbances
1193.           4.4. Operations
1194.           4.5. Gate Actuators
1195.           4.6. Turbine Actuators
1196.
1197.       """
1198.
1199.       """
1200.       4.1. HYDRAULIC SYSTEM STATE
1201.       """
1202.
1203.       initial_reservoir_storage=SSCrev(B_RSEs[0,year])
1204.       if initial_reservoir_storage<=-
    304.1:  #making sure initial reservoir level isn't breach level, if any
1205.           initial_reservoir_storage=364.27
1206.
1207.       @sdpy.stock(model, initial_reservoir_storage, name='Reservoir Storage',c
    ache=False, jit=False)
1208.       def reservoir_storage(t):
1209.           out=reservoir_inflow(t) - reservoir_outflow(t)
1210.           return out
1211.
1212.       @sdpy.aux(model, name='Inflow',cache=False, jit=False)
1213.       def reservoir_inflow(t):
1214.           daytimestep=t
1215.           return float(Inflow[daytimestep,year])
```

```
1216.
1217.      @sdpy.aux(model, name="Outflow",cache=False, jit=False)
1218.      def reservoir_outflow(t):
1219.          return float(gated_spill_release(t) + overtopping_flow(t) + power_fl
    ow_release(t) + penstock_leakage(t) + earth_dam_seepage(t) + breach_flow(t))
1220.
1221.      @sdpy.aux(model, name="Reservoir Level",cache=False, jit=False)
1222.      def reservoir_level(t):
1223.          return SSC(reservoir_storage(t))
1224.
1225.      @sdpy.aux(model, name="Overtopping Flow",cache=False, jit=False)
1226.      def overtopping_flow(t):
1227.          level=reservoir_level(t)
1228.          storage=reservoir_storage(t)
1229.          pf=power_flow_release(t)
1230.          sf=gated_spill_release(t)
1231.          inf=reservoir_inflow(t)
1232.          storage2=storage+inf-pf-sf
1233.          Overtoppingflow=0
1234.          OTs=np.zeros(24)
1235.          if level>378.41 or SSC(storage2)>378.41:
1236.              for i in range(24):
1237.                  OTs[i]=max(OTC(SSC(storage)), 0)*(1/24.)
1238.                  #water balance
1239.                  storage=storage+inf/24.-OTs[i]-pf/24.-sf/24.
1240.              Overtoppingflow=np.sum(OTs)
1241.          return Overtoppingflow
1242.
1243.      @sdpy.aux(model, name="Unobstructed Gate Flow",cache=False, jit=False)
1244.      def unobstructed_gate_flow(t):
1245.      #    av=gate_availability(t)
1246.      #    ops=operations_planning(t)[0]
1247.      #    SC=sensor_condition(t)
1248.      #    Se=sensor_error(t)
1249.          level=reservoir_level(t)
1250.      #    SE=1
1251.      #    Spillflow=0
1252.      #    if Se!=0:
1253.      #        SE=0
1254.      #    if av==1 and (SE+SC==2) and level>367.28:
1255.      #        #if gate available, sensors functional, man act working or some
    one on site set directly to operations plan
1256.      #        Spillflow=ops
1257.      #    else:
1258.              #if gate unavailable or issues with actuation or sensors, use ga
    te position to determine flow
1259.          g = gate_position(t)
1260.          Spillflow=GateFlowCalc(level,g)
1261.          return Spillflow
1262.
1263.      @sdpy.aux(model, name="Gated Spill Release",cache=False, jit=False)
1264.      def gated_spill_release(t):
1265.          if breach_triggered(t)==0:
1266.              if components_collapsing_gate(t)==1:
1267.                  return gate_capacity(t)*unobstructed_gate_flow(t)
1268.              else:
1269.                  return min(1590, max(reservoir_storage(t) + reservoir_inflow
    (t) - 83.1357, 0))
1270.          else:
1271.              return 0
1272.
```

```
1273.       DebrisRemoval= np.zeros(1)
1274.       @sdpy.aux(model, name="Gated Capacity",cache=False, jit=False)
1275.       def gate_capacity(t):
1276.           timestep=t
1277.           #DebrisRemoval = 0 #global todo
1278.           currentinflow=reservoir_inflow(t)
1279.           inflowthreshold=65 #assume less than this required to remove debris

1280.           Capacity=1
1281.           if ReducedCapacities[year]<1:
1282.               if timestep>=1 and timestep<=24.*ReducedCapacityMinimumTime:
1283.                   Capacity=ReducedCapacities[year]
1284.               if timestep>=24.*ReducedCapacityMinimumTime: #debris removal can
     start after a minimum time
1285.                   if DebrisRemoval[0]==0:
1286.                       if currentinflow<inflowthreshold:  #Debris removal only
    less than inflow threshold
1287.                           DebrisRemoval[0]=1
1288.                   if DebrisRemoval[0]==0:  #if debrs, set to reduced capacitie
    s
1289.                       Capacity=ReducedCapacities[year]
1290.                   if DebrisRemoval[0]==1: #if debris removed, set to full capa
    city
1291.                       Capacity=1
1292.           return float(Capacity)
1293.
1294.       @sdpy.aux(model, name="Power Flow Release",cache=False, jit=False)
1295.       def power_flow_release(t):
1296.           if breach_triggered(t)==0:
1297.               return powerhouse_flow_conveyance(t)
1298.           else:
1299.               return 0
1300.
1301.       @sdpy.aux(model, name="Penstock Leakage",cache=False, jit=False)
1302.       def penstock_leakage(t):
1303.           if intake_gate_closure(t)==0 and breach_triggered(t)==0 and other_co
    mponent_remaining_time_to_repair(t)[1]>0:
1304.               return head_cover_max_flow(t)
1305.           else:
1306.               return 0
1307.
1308.       RESEL_IG=[]
1309.       @sdpy.aux(model, name="Intake Gate Closure",cache=False, jit=False)
1310.       def intake_gate_closure(t):
1311.           penstockrup=other_component_remaining_time_to_repair(t)[1]
1312.           hcfail=power_remaining_time_to_repair(t)[0]
1313.           igclosed=0
1314.           if hcfail>0 or penstockrup>0:
1315.               RESEL_IG.append(reservoir_level(t))
1316.               if (np.min(RESEL_IG)<363.06): #Intake gate can be closed once re
    servoir drawn down past sill of intake gate
1317.                   igclosed=1
1318.           return igclosed
1319.
1320.       @sdpy.aux(model, name="Uncontrolled Release",cache=False, jit=False)
1321.       def uncontrolled_release(t):
1322.           ucr=0
1323.           if head_cover(t)==0:
1324.               ucr+=np.max([power_flow_release(t),0])
1325.           if gate_collapse(t)==1:
1326.               ucr+=np.max([gated_spill_release(t),0])
```

```
1327.           ucr+=np.max([penstock_leakage(t),0])
1328.           ucr+=np.max([breach_flow(t),0])
1329.           ucr+=overtopping_flow(t)
1330.           return ucr
1331.
1332.       BREACHT=np.zeros((365,NYr))
1333.       @sdpy.aux(model, name="Breach Triggered",cache=False, jit=False)
1334.       def breach_triggered(t):
1335.           bt=np.max(RSEs[:,year])
1336.           if bt>381.73:
1337.               BREACHT[t,year]=1
1338.               return 1
1339.           else:
1340.               return 0
1341.
1342.       @sdpy.aux(model, name="Breach Flow",cache=False, jit=False)
1343.       def breach_flow(t):
1344.           if breach_triggered(t)==1:
1345.               return reservoir_storage(t) + reservoir_inflow(t) - (-304.012)
1346.           else:
1347.               return 0
1348.
1349.       @sdpy.aux(model, name="Earth Dam Settlement",cache=False, jit=False)
1350.       def earth_dam_settlement(t):
1351.           return 0 #not used for this model
1352.
1353.       @sdpy.aux(model, name="Earth Dam Seepage",cache=False, jit=False)
1354.       def earth_dam_seepage(t):
1355.           return 0 #not used for this model
1356.           #IF THEN ELSE(Earth dam settlement=0, 0 ,
1357.       #IF THEN ELSE(Reservoir Level>364.9, Earth dam settlement*Reservoir Leve
    l*0.1 , 0 ))
1358.
1359.
1360.       """
1361.       4.2. SENSORS
1362.
1363.       """
1364.
1365.       if seedgen==0:
1366.           SErrorDeltat=Seeds['SErrorDeltat']
1367.           SErrorDuration=Seeds['SErrorDuration']
1368.           SErrors=Seeds['SErrors']
1369.
1370.       @sdpy.aux(model, name="Sensor Condition",cache=False, jit=False)
1371.       def sensor_condition(t):
1372.           if sensor_remaining_time_to_repair(t)>0:
1373.               return 0
1374.           else:
1375.               return 1
1376.
1377.       @sdpy.aux(model, name="Gauge Reading",cache=False, jit=False)
1378.       def gauge_reading(t):
1379.           if sensor_condition(t)==1:
1380.               return reservoir_level(t) + (sensor_error(t)/100)*reservoir_leve
    l(t)
1381.           else:
1382.               return -1000
1383.           #IF THEN ELSE( Sensor condition=1 , Reservoir Level+((Sensor Error)/
    100)*Reservoir Level , -1000)
1384.
```

```
1385.       @sdpy.aux(model, name="Gauge Processing",cache=False, jit=False)
1386.       def gauge_processing(t):
1387.           if other_component_remaining_time_to_repair(t)[0]>0:
1388.               return -1000
1389.           else:
1390.               return gauge_reading(t)
1391.           #IF THEN ELSE(Other component remaining time to repair[PLCRTU]>0, -
    1000 , Gauge reading)
1392.
1393.       @sdpy.aux(model, name="Gauge Relay",cache=False, jit=False)
1394.       def gauge_relay(t):
1395.           return gauge_processing(t)
1396.
1397.       @sdpy.aux(model, name="Sensor Error",cache=False, jit=False)
1398.       def sensor_error(t):
1399.           timestep=t
1400.           error=0
1401.           if (timestep>=SErrorDeltat[year] and timestep<=(SErrorDeltat[year]+S
    ErrorDuration[year])):
1402.               error=SErrors[year] #sets sensor components to failure time
1403.           return error
1404.
1405.       """
1406.       4.3 DISTURBANCES
1407.
1408.       """
1409.
1410.       gatecomps = sdpy.SubRange('gatecomps', ['C_FC', 'C_FO', 'C_FIP'])
1411.       turbinecomps = sdpy.SubRange('turbinecomps', ['HC', 'GEN'])
1412.       othercomps = sdpy.SubRange('othercomps', ["PLCRTU", "PEN", "GRID"])
1413.
1414.
1415.       if seedgen==0:
1416.           GateOutagesAll=Seeds['GateOutagesAll']
1417.           GateCollapses=Seeds['GateCollapses']
1418.           Gdeltat=Seeds['Gdeltat']
1419.
1420.       @sdpy.aux(model, name="Gate Component Failures",cache=False, jit=False)

1421.       @sdpy.subscript(gatecomps)
1422.       def gate_component_failures(t):
1423.           timestep=t
1424.           timetorepair=np.zeros(3)
1425.           #SPOG1, 3 different general components can be set to failure, also o
    ne general
1426.           #C_FC, C_FO, C_FIP (fail open/collapse, fail closed, fail in place)

1427.           for c in range(3):
1428.               if timestep==Gdeltat[c, year]:
1429.                   if GateOutagesAll[c, year]>0:
1430.                       timetorepair[c]=GateOutagesAll[c, year] #sets gate compo
    nents to failure time
1431.           return timetorepair
1432.
1433.       @sdpy.aux(model, name='Gate Time To Repair',cache=False, jit=False)
1434.       @sdpy.subscript(gatecomps)
1435.       def gate_time_to_repair(t):
1436.           return gate_component_failures(t)
1437.
1438.       @sdpy.aux(model, name='Gate Repair',cache=False, jit=False)
1439.       @sdpy.subscript(gatecomps)
```

```
1440.       def gate_repair(t):
1441.           ret=np.zeros(3)
1442.           for i in range(3):
1443.               if gate_remaining_time_to_repair(t)[i]>1:
1444.                   ret[i]=1
1445.               else:
1446.                   if gate_remaining_time_to_repair(t)[i]<=1 and gate_remaining
    _time_to_repair(t)[i]>0:
1447.                       ret[i]=gate_remaining_time_to_repair(t)[i]
1448.                   else:
1449.                       ret[i]=0
1450.           return ret
1451.       #if_then_else(gate_remaining_time_to_repair(t)>1, 1,
1452.       #             if_then_else(gate_remaining_time_to_repair(t)<1 and gate_r
    emaining_time_to_repair(t)>0, gate_remaining_time_to_repair(t),0) )
1453.       #    IF THEN ELSE(Gate remaining time to repair[GateComps]>1, 1 , IF THE
    N ELSE(Gate remaining time to repair[GateComps]<1 :AND: Gate remaining time to r
    epair[GateComps]>0,Gate remaining time to repair[GateComps],0) )
1454.
1455.       @sdpy.stock(model, np.zeros(3), name='Gate Remaining Time To Repair',cac
    he=False, jit=False)
1456.       @sdpy.subscript(gatecomps)
1457.       def gate_remaining_time_to_repair(t):
1458.           return gate_time_to_repair(t) - gate_repair(t)
1459.
1460.       GFORTTR=np.zeros((365,NYr))
1461.       @sdpy.aux(model, name='Gate All',cache=False, jit=False)
1462.       def gate_all(t):
1463.           GFORTTR[t,year]=gate_remaining_time_to_repair(t)[1]
1464.           maxs=np.max(gate_remaining_time_to_repair(t))
1465.           return maxs
1466.
1467.       if seedgen==0:
1468.           TurbineOutagesAll=Seeds['TurbineOutagesAll']
1469.           Tdeltat=Seeds['Tdeltat']
1470.
1471.       @sdpy.aux(model, name='Power component Failures',cache=False, jit=False)

1472.       @sdpy.subscript(turbinecomps)
1473.       def power_component_failures(t):
1474.           timestep=t
1475.           timetorepair=np.zeros(2)   #gen #HC
1476.           for c in range(2):
1477.               if timestep==Tdeltat[c, year]:
1478.                   if TurbineOutagesAll[c, year]>0:
1479.                       timetorepair[c]=TurbineOutagesAll[c, year] #sets gate co
    mponents to failure time
1480.           return timetorepair
1481.
1482.       @sdpy.aux(model, name='Power Time To Repair',cache=False, jit=False)
1483.       @sdpy.subscript(turbinecomps)
1484.       def power_time_to_repair(t):
1485.           return power_component_failures(t)
1486.
1487.       @sdpy.aux(model, name='Power Repair',cache=False, jit=False)
1488.       @sdpy.subscript(turbinecomps)
1489.       def power_repair(t):
1490.           ret=np.zeros(2)
1491.           for i in range(2):
1492.               if power_remaining_time_to_repair(t)[i]>1:
1493.                   ret[i]=1
```

```
1494.             else:
1495.                 if power_remaining_time_to_repair(t)[i]<=1 and power_remaini
    ng_time_to_repair(t)[i]>0:
1496.                     ret[i]=power_remaining_time_to_repair(t)[i]
1497.                 else:
1498.                     ret[i]=0
1499.         return ret
1500.     #    if_then_else(power_remaining_time_to_repair(t)>1, 1,
1501.     #             if_then_else(power_remaining_time_to_repair(t)<1 and power
    _remaining_time_to_repair(t)>0, power_remaining_time_to_repair(t),0) )
1502.
1503.     @sdpy.stock(model, np.zeros(2), name='Power Remaining Time To Repair',ca
    che=False, jit=False)
1504.     @sdpy.subscript(turbinecomps)
1505.     def power_remaining_time_to_repair(t):
1506.         return power_time_to_repair(t) - power_repair(t)
1507.
1508.     @sdpy.aux(model, name='Power All',cache=False, jit=False)
1509.     def power_all(t):
1510.         maxs=np.max(power_remaining_time_to_repair(t))
1511.         return maxs
1512.
1513.     if seedgen==0:
1514.         OCdeltat=Seeds['OCdeltat']
1515.         OCOutages=Seeds['OCOutages']
1516.
1517.     @sdpy.aux(model, name='Other Component Failures',cache=False, jit=False)

1518.     @sdpy.subscript(othercomps)
1519.     def other_component_failures(t):
1520.         timetorepair=np.zeros(3)
1521.         #0 Dam PLCRTU, 1 Penstock, 2 Grid
1522.         timestep=t
1523.         for c in range(3):
1524.             if timestep==OCdeltat[c, year]:
1525.                 if OCOutages[c, year]>0:
1526.                     timetorepair[c]=OCOutages[c, year] #sets other component
    s to failure time
1527.
1528.         return timetorepair
1529.
1530.     @sdpy.aux(model, name='Other Component Time To Repair',cache=False, jit=
    False)
1531.     @sdpy.subscript(othercomps)
1532.     def other_component_time_to_repair(t):
1533.         return other_component_failures(t)
1534.
1535.     @sdpy.aux(model, name='Other Component Repair',cache=False, jit=False)
1536.     @sdpy.subscript(othercomps)
1537.     def other_component_repair(t):
1538.         ret=np.zeros(3)
1539.         for i in range(3):
1540.             if other_component_remaining_time_to_repair(t)[i]>1:
1541.                 ret[i]=1
1542.             else:
1543.                 if other_component_remaining_time_to_repair(t)[i]<=1 and oth
    er_component_remaining_time_to_repair(t)[i]>0:
1544.                     ret[i]=other_component_remaining_time_to_repair(t)[i]
1545.                 else:
1546.                     ret[i]=0
1547.         return ret
```

```
1548.       #    if_then_else(other_component_remaining_time_to_repair(t)>1, 1,
1549.       #             if_then_else(other_component_remaining_time_to_repair(t)<1
    and other_component_remaining_time_to_repair(t)>0, other_component_remaining_ti
    me_to_repair(t),0) )
1550.
1551.       @sdpy.stock(model, np.zeros(3), name='Other Component Remaining Time To
    Repair',cache=False, jit=False)
1552.       @sdpy.subscript(othercomps)
1553.       def other_component_remaining_time_to_repair(t):
1554.           return other_component_time_to_repair(t)-other_component_repair(t)
1555.
1556.       if seedgen==0:
1557.           SOutages=Seeds['SOutages']
1558.           Sdeltat=Seeds['SOutages']
1559.
1560.       @sdpy.aux(model, name='Sensor Failures',cache=False, jit=False)
1561.       def sensor_failures(t):
1562.           timestep=t
1563.           timetorepair=0
1564.           if timestep==Sdeltat[year]:
1565.               if SOutages[year]>0:
1566.                   timetorepair=SOutages[year] #sets sensor components to failu
    re time
1567.           return timetorepair
1568.
1569.       @sdpy.aux(model, name='Sensor Time To Repair',cache=False, jit=False)
1570.       def sensor_time_to_repair(t):
1571.           return sensor_failures(t)
1572.
1573.       @sdpy.aux(model, name='Sensor Repair',cache=False, jit=False)
1574.       def sensor_repair(t):
1575.           if sensor_remaining_time_to_repair(t)>1:
1576.               return 1
1577.           else:
1578.               if sensor_remaining_time_to_repair(t)<=1 and sensor_remaining_ti
    me_to_repair(t)>0:
1579.                   return sensor_remaining_time_to_repair(t)
1580.               else:
1581.                   return 0
1582.
1583.       @sdpy.stock(model, 0, name='Sensor Remaining Time To Repair',cache=False
    , jit=False)
1584.       def sensor_remaining_time_to_repair(t):
1585.           return sensor_time_to_repair(t)-sensor_repair(t)
1586.
1587.
1588.       """
1589.       4.4. OPERATIONS
1590.
1591.       """
1592.       controls = sdpy.SubRange('controls', ['g1', 't1'])
1593.
1594.       if seedgen==0:
1595.           IFErrorDeltat=Seeds['IFErrorDeltat']
1596.           IFErrorDuration=Seeds['IFErrorDuration']
1597.
1598.
1599.       @sdpy.aux(model, name='Operations Planning',cache=False, jit=False)
1600.       @sdpy.subscript(controls)
1601.       def operations_planning(t):
1602.           timestep=t
```

```
1603.           dayref=dayrefs(Startdays[year], timestep)
1604.           FishFlow=getfishflow(dayref)
1605.           Inf114=Inflow[timestep:14+timestep, year] #Changed to one day ahead
    so proper spills are released for Vensim version
1606.           InfForecast=Inf114
1607.           gaugerelay=gauge_relay(t)
1608.           if gaugerelay<-
    900 or gaugerelay>381.73: #If error is so high that it becomes obvious
1609.               storage=-1000
1610.           else:
1611.               storage=SSCrev(gaugerelay)
1612.           StaffOnSite=site_staff_mobilized(t)
1613.           actualstorage=reservoir_storage(t)
1614.           if (StaffOnSite>0) or (storage==-1000):
1615.               if storage==-1000:
1616.                   storage=SSCrev(RSEs[timestep-
    1, year]) #if unknown, takes previous days value
1617.               if StaffOnSite==1:
1618.                   storage=actualstorage  #if someone is on site, takes actual
    value
1619.           InitialStorage=np.float64(storage)#+lastinf-outfs
1620.           resElPens=np.zeros((3,3))#Penalties for res el
1621.           resElPens[0,:]=[0,273,304]
1622.           resElPens[1,:]=[426.99, 300.39, 300.39]
1623.           resElPens[2,:]=[99.58693574984267,99.58693574984267, 171.28] #123.60
    856547318923 from 87.055 to help reduce 0 spill events
1624.           penstockrup=other_component_remaining_time_to_repair(t)[1] #penstock

1625.           hcfail=power_remaining_time_to_repair(t)[0] #head cover
1626.           igate=intake_gate_closure(t)
1627.           if penstockrup>0 or hcfail>0:
1628.               if igate==0: #reduce res el targets to get res below intake gate
     sill so it can be closed
1629.                   resElPens[1,:]=[-48.5, -48.5, -48.5]
1630.                   resElPens[2,:]=[-47, -47, -47] #lowered
1631.               #draw down reservoir to sill
1632.           SPOG1Av=gate_all(t)   #Availbility set based on "gate time to repair
    "
1633.           TurbAv1=power_all(t)
1634.           Nextday=np.zeros(2)
1635.
1636.           Optimized=OpsPlan(InfForecast, InitialStorage, dayref,  SPOG1Av,  Tu
    rbAv1,  resElPens)
1637.           Nextday=np.array(Optimized.copy())   #SPOG1, Turb1
1638.
1639.           Nextday.clip(min=0) #omit negatives.
1640.
1641.           return Nextday
1642.
1643.       BREACHQ=np.zeros((365, NYr))
1644.       IGCLOSE=np.zeros((365, NYr))
1645.       @sdpy.aux(model, name='Output Saving',cache=False, jit=False)
1646.       def output_saving(t):
1647.           timestep=t
1648.           outfs=reservoir_outflow(t)
1649.           IGCLOSE[timestep,year]=intake_gate_closure(t)
1650.           BREACHQ[timestep,year]=breach_flow(t)
1651.           GAVs[timestep, year]=unit_availability(t)
1652.           UAVs[timestep,year]=gate_availability(t)
1653.           timetorepairg=gate_remaining_time_to_repair(t) #model['Gate remainin
    g time to repair[GateComps]']
```

```
1654.           timetorepairt=power_remaining_time_to_repair(t) #model['Power remain
    ing time to repair[TurbineComps]']
1655.           RSEs[timestep, year]=reservoir_level(t) #model['Reservoir Level']
1656.           TBFs[timestep, year]=power_flow_release(t) #model["Power flow releas
    e"]
1657.           SPOGs[timestep, year]=gated_spill_release(t) #model["Gated spill rel
    ease"]
1658.           OT[timestep, year]=overtopping_flow(t)
1659.           INFs[timestep,year]=reservoir_inflow(t) #model["Inflow"]
1660.           OUTFs[timestep,year]=outfs
1661.           TTRS[timestep,0:3, year]=gate_remaining_time_to_repair(t) #model['Ga
    te remaining time to repair[GateComps]']
1662.           TTRS[timestep,3:5, year]=power_remaining_time_to_repair(t) #model['P
    ower remaining time to repair[TurbineComps]']
1663.           TTRS[int(timestep),5:8, year]=other_component_remaining_time_to_repa
    ir(t) #model['Other component remaining time to repair[Other infrastructure]']
1664.           if TTRS[timestep, 2, year]<=0:
1665.               GPs[timestep,year]=gate_position(t) #model["Gate Position"]
1666.           if TTRS[timestep, 2, year]>0:
1667.               GPs[timestep,year]=GPs[timestep-1, year]
1668.           AllMaxQ_t[timestep,:, year]=np.array(AllMaxQ)
1669.           if timetorepairg[0]>0:
1670.               AllMaxQ_t[timestep,0, year] = 0 #gate fails closed, cap is at 0

1671.           if timetorepairg[1]>0:
1672.               AllMaxQ_t[timestep,0, year] = 861.1+728.9 #gate fails open, cap
    is maxed
1673.           if timetorepairg[2]>0:
1674.               tempgatecap=GateFlowCalc(381.6,gate_position(t))
1675.               AllMaxQ_t[timestep,0, year] = tempgatecap #gate fails in place,
    cap is max flow @ current opening
1676.           if intake_gate_closure(t)==1: #turbine capacity is zero when intake
    gate closed
1677.               AllMaxQ_t[timestep, 1, year]=0
1678.           GateCaps[t,year]=gate_capacity(t)
1679.           AllMaxQ_t[timestep,0, year]=np.multiply(AllMaxQ_t[int(timestep),0, y
    ear], GateCaps[t,year]) #account for debris blockage
1680.           if timetorepairt[1]>0: #Generator outage
1681.                AllMaxQ_t[timestep,1, year]=0
1682.           CAPs[timestep, year]=np.sum(AllMaxQ_t[timestep,:, year])
1683.           UCRs[timestep, year]=uncontrolled_release(t)
1684.           GCRs[timestep, year]=gate_control_redundancy(t)
1685.           return 1
1686.
1687.
1688.       @sdpy.aux(model, name='Turbine Instructions',cache=False, jit=False)
1689.       def turbine_instructions(t):
1690.           return operations_planning(t)[1]
1691.
1692.
1693.       @sdpy.aux(model, name='Gate Instructions',cache=False, jit=False)
1694.       def gate_instructions(t):
1695.           ResEl=reservoir_level(t)
1696.           OP=operations_planning(t)[0]
1697.           gps=GateInstr(ResEl, OP)
1698.       #    print("getALLgp: Gate Instruction q: "+str(OPs[0]) + "  Gate positi
    on: " +str(gps) + "  Res El: "+str(ResEl))
1699.           return gps
1700.
1701.       @sdpy.aux(model, name='Manual Actuation',cache=False, jit=False)
1702.       def manual_actuation(t):
```

```
            if other_component_remaining_time_to_repair(t)[0]>0 or dam_grid_avai
    lability(t)==0 or sensor_remaining_time_to_repair(t)>0:
                MOBI[t,year]=1
                return 1
            else:
                return 0
        #IF THEN ELSE(Other component remaining time to repair[PLCRTU]>0 :OR: Ga
    te instructions>2 :OR: Dam grid availability=0 :OR: Sensor remaining time to rep
    air>0, 1 , 0 )+0*Operations planning[g1]

        @sdpy.aux(model, name='Initiate',cache=False, jit=False)
        def initiate(t):
            #IF THEN ELSE(Manual actuation=1:AND:Manual Actuation Initiated<=0,
    1 , 0 )
            if manual_actuation(t)==1 and manual_actuation_initiated(t)<=0:
                return 1
            else:
                return 0

        @sdpy.aux(model, name='Demobilize',cache=False, jit=False)
        def demobilize(t):
            #IF THEN ELSE(Manual actuation=0:AND:Site staff mobilized=1, 1 , 0)

            os=output_saving(t)
            if manual_actuation(t)==0 and site_staff_mobilized(t)==1:
                return 1
            else:
                return 0

        @sdpy.stock(model, 0, name='Manual Actuation Initiated',cache=False, jit
    =False)
        def manual_actuation_initiated(t):
            return initiate(t)-demobilize(t)

        if seedgen==0:
            DelayStaff=Seeds['DelayStaff']

        @sdpy.aux(model, name='Delay In Contacting Staff',cache=False, jit=False
    )
        def delay_in_contacting_staff(t):
            delay=1
            if manual_actuation_initiated(t)==1:
                delay=DelayStaff[year] #sets sensor components to failure time
            return float(delay)

        if seedgen==0:
            DelayAccess=Seeds["DelayAccess"]

        @sdpy.aux(model, name='Delay In Accessing Site',cache=False, jit=False)

        def delay_in_accessing_site(t):
            delay=3+np.zeros(2)
            for c in range(2):
                if manual_actuation_initiated(t)==1:
                        delay[c]=DelayAccess[c, year]
            return float(delay[0]) #Can add powerhouse delays later, ignoring fo
    r now.

        @sdpy.aux(model, name='Contact Initiation',cache=False, jit=False)
        def contact_initiation(t):
```

```
1754.           #IF THEN ELSE(Initiate=1 :AND: Site staff mobilized=0 :AND: Plant st
    aff notified=0, Delay in contacting staff
1755.       #, IF THEN ELSE(Demobilize=1, 1 , 0 ))
1756.           if initiate(t)==1 and site_staff_mobilized(t)==0 and plant_staff_not
    ified(t)==0:
1757.               return delay_in_contacting_staff(t)
1758.           else:
1759.               if demobilize(t)==1:
1760.                   return 1
1761.               else:
1762.                   return 0
1763.
1764.       @sdpy.aux(model, name='Contacting',cache=False, jit=False)
1765.       def contacting(t):
1766.           #IF THEN ELSE(Time remaining to contact plant manager and site staff
    >0 :AND: Manual Actuation Initiated=1, IF THEN ELSE(Time remaining to contact pl
    ant manager and site staff<1, Time remaining to contact plant manager and site s
    taff, 1) , 0 )
1767.           if time_remaining_to_contact_staff(t)>0 and manual_actuation_initiat
    ed(t)==1:
1768.               if time_remaining_to_contact_staff(t)<1:
1769.                   return time_remaining_to_contact_staff(t)
1770.               else:
1771.                   return 1
1772.           else:
1773.               return 0
1774.
1775.       @sdpy.stock(model, 1, name='Time Remaining To Contact Staff',cache=False
    , jit=False)
1776.       def time_remaining_to_contact_staff(t):
1777.           return contact_initiation(t)-contacting(t)
1778.
1779.       @sdpy.aux(model, name='Plant Staff Notified',cache=False, jit=False)
1780.       def plant_staff_notified(t):
1781.           #IF THEN ELSE(Time remaining to contact plant manager and site staff
    <=0 :AND: Manual Actuation Initiated=1, 1 , 0 )
1782.           if time_remaining_to_contact_staff(t)<=0 and manual_actuation_initia
    ted(t)==1:
1783.               return 1
1784.           else:
1785.               return 0
1786.
1787.       plantStaffNotified=np.zeros(365)
1788.       @sdpy.aux(model, name='Mobilization Initiated',cache=False, jit=False)
1789.       def mobilization_initiated(t):
1790.           time=t
1791.           plantStaffNotified[int(time)]=plant_staff_notified(t)
1792.           AccessDelay=delay_in_accessing_site(t)
1793.           demob=demobilize(t)
1794.           Mobinit=0
1795.           if time>0:
1796.               if plantStaffNotified[int(time)]==1 and plantStaffNotified[int(t
    ime-1)]==0:
1797.                   Mobinit=AccessDelay  #Adding delays in access time to stock

1798.           if demob==1:
1799.               Mobinit=1 #returning stock to demobilized value which is 3 hr ti
    me to get to site on av
1800.           return float(Mobinit)
1801.
1802.
```

```
1803.      @sdpy.aux(model, name='Mobilizing',cache=False, jit=False)
1804.      def mobilizing(t):
1805.          #IF THEN ELSE(Time remaining to access site>0 :AND: Manual Actuation
     Initiated=1 :AND: Time remaining to contact plant manager and site staff<=0, IF
     THEN ELSE(Time remaining to access site<1, Time remaining to access site, 1), 0
     )
1806.          if time_remaining_to_access_site(t)>0 and manual_actuation_initiated
    (t)==1 and time_remaining_to_contact_staff(t)<=0:
1807.              if time_remaining_to_access_site(t)<1:
1808.                  return time_remaining_to_access_site(t)
1809.              else:
1810.                  return 1
1811.          else:
1812.              return 0
1813.
1814.      @sdpy.stock(model, 1, name='Time Remaining To Access Site',cache=False,
    jit=False)
1815.      def time_remaining_to_access_site(t):
1816.          return mobilization_initiated(t)-mobilizing(t)
1817.
1818.      @sdpy.aux(model, name='Site Staff Mobilized',cache=False, jit=False)
1819.      def site_staff_mobilized(t):
1820.          #IF THEN ELSE(Time remaining to access site<=0 :AND: Manual Actuatio
    n Initiated=1 :AND: Time remaining to contact plant manager and site staff<=0, 1
    , 0)
1821.          if time_remaining_to_access_site(t)<=0 and manual_actuation_initiate
    d(t)==1 and time_remaining_to_contact_staff(t)<=0:
1822.              MOB[t,year]=1
1823.              return 1
1824.          else:
1825.              return 0
1826.
1827.      @sdpy.aux(model, name='Gate Control Redundancy',cache=False, jit=False)

1828.      def gate_control_redundancy(t):
1829.          #IF THEN ELSE(Manual actuation=0, 2, IF THEN ELSE(Manual actuation=1
     :AND: Site staff mobilized=1, 1 , 0 ))
1830.          if manual_actuation(t)==0:
1831.              return 2
1832.          else:
1833.              if manual_actuation(t)==1 and site_staff_mobilized(t)==1:
1834.                  return 1
1835.              else:
1836.                  return 0
1837.
1838.      """
1839.      4.5.  GATE ACTUATORS
1840.
1841.      """
1842.
1843.      @sdpy.aux(model, name='Dam Grid Availability',cache=False, jit=False)
1844.      def dam_grid_availability(t):
1845.          #IF THEN ELSE(Other component remaining time to repair[Grid]>0, 0, 1
    )
1846.          if other_component_remaining_time_to_repair(t)[2]>0:
1847.              return 0
1848.          else:
1849.              return 1
1850.
1851.      @sdpy.aux(model, name='Gate Power Supply',cache=False, jit=False)
1852.      def gate_power_supply(t):
```

```
            return dam_grid_availability(t)

        @sdpy.aux(model, name="Gate Availability",cache=False, jit=False)
        def gate_availability(t):
            time=t
            gateavail=1
            sstaff=site_staff_mobilized(t)
            staffproblemgate=0
            if time>0:
                if (other_component_remaining_time_to_repair(t)[0]>0 or other_co
    mponent_remaining_time_to_repair(t)[2]>0) and sstaff==0:
                    staffproblemgate=1
                gateout=gate_all(t)
                if gateout>0 or staffproblemgate==1:
                    gateavail=0
            return float(gateavail)

        @sdpy.aux(model, name="Components Failing Gate In Place",cache=False, ji
    t=False)
        def components_failing_gate_in_place(t):
            #IF THEN ELSE(Gate remaining time to repair[C FIP]>0, 0 , 1 )
            if gate_remaining_time_to_repair(t)[2]>0:
                return 0
            else:
                return 1

        @sdpy.aux(model, name="Components Collapsing Gate",cache=False, jit=Fals
    e)
        def components_collapsing_gate(t):
            #IF THEN ELSE(Gate remaining time to repair[C FO]>0, 0 , 1 )
            if gate_remaining_time_to_repair(t)[1]>0:
                return 0
            else:
                return 1

        @sdpy.aux(model, name="Components Failing Gate Closed",cache=False, jit=
    False)
        def components_failing_gate_closed(t):
            #IF THEN ELSE(Gate remaining time to repair[C FC]>0, 0 , 1 )
            if gate_remaining_time_to_repair(t)[0]>0:
                return 0
            else:
                return 1

        @sdpy.aux(model, name="Gate Collapsed",cache=False, jit=False)
        def gate_collapse(t):
            #IF THEN ELSE(Components collapsing gate=0, 1 , 0 )
            if components_collapsing_gate(t)==0:
                return 1
            else:
                return 0

        @sdpy.aux(model, name="Fail Closed",cache=False, jit=False)
        def fail_closed(t):
            #IF THEN ELSE(Components failing gate closed=0, 1 , 0)
            if components_failing_gate_closed(t)==0:
                return 1
            else:
                return 0

        max_opening=12.5
```

```
1910.
1911.       @sdpy.aux(model, name="Last Gate Position",cache=False, jit=False)
1912.       def last_gate_position(t):
1913.           timestep=t
1914.           gp=GPs[timestep,year]
1915.           if gate_availability(t)==0:
1916.               gateavinds=np.where(GAVs[:,year]==1)[0]
1917.               if np.size(gateavinds)!=0:
1918.                   lastgateactivets=np.max(gateavinds)
1919.                   gp=GPs[lastgateactivets, year]
1920.           return gp
1921.
1922.       @sdpy.aux(model, name="Gate Position",cache=False, jit=False)
1923.       def gate_position(t):
1924.           GColl=gate_collapse(t)
1925.           MaxO=12.5
1926.           GFClosed=fail_closed(t)
1927.           GateAvailability=gate_availability(t)
1928.           GAVs[int(t), year]=GateAvailability
1929.           GateInstructions=gate_instructions(t)
1930.           LastGatePosition=last_gate_position(t)
1931.           if GColl==1:
1932.               return MaxO
1933.           if GFClosed==1:
1934.               return 0
1935.           if GateAvailability==1:
1936.               return GateInstructions
1937.           if GateAvailability==0:
1938.               return LastGatePosition
1939.
1940.
1941.       """
1942.       4.6. TURBINE ACTUATORS
1943.
1944.       """
1945.
1946.       @sdpy.aux(model, name='Unit Availability',cache=False, jit=False)
1947.       def unit_availability(t):
1948.           time=t
1949.           turbavail=1
1950.           if time>0:
1951.               turbout=power_all(t)
1952.               penstockrup=other_component_remaining_time_to_repair(t)[1] #pens
    tock
1953.               if turbout>0:
1954.                   turbavail=0
1955.               if penstockrup>0:
1956.                   turbavail=0
1957.               if other_component_remaining_time_to_repair(t)[2]>0: #grid failu
    re
1958.                   turbavail=0
1959.           return turbavail
1960.
1961.       @sdpy.aux(model, name='Head Cover',cache=False, jit=False)
1962.       def head_cover(t):
1963.           if power_remaining_time_to_repair(t)[0]>0:
1964.               return 0
1965.           else:
1966.               return 1
1967.
1968.       @sdpy.aux(model, name='Generator',cache=False, jit=False)
```

```
1969.      def generator(t):
1970.          if power_remaining_time_to_repair(t)[1]>0:
1971.              return 0
1972.          else:
1973.              return 1
1974.
1975.      @sdpy.aux(model, name='Head Cover Max Flow',cache=False, jit=False)
1976.      def head_cover_max_flow(t):
1977.          flag1=head_cover(t)+1 #if head cover = 0 then flag=1
1978.          if other_component_remaining_time_to_repair(t)[1]>0: #penstock
1979.              flag1=1
1980.          resels=reservoir_level(t)
1981.          t1=fncTurbineMaxFlow(resels, flag1)
1982.          if intake_gate_closure(t)==1:
1983.              return 0
1984.          if intake_gate_closure(t)==0:
1985.              return np.max([np.min([5*(t1), reservoir_storage(t)+reservoir_in
    flow(t)-gated_spill_release(t)-(-48.6)]), 0])
1986.
1987.      @sdpy.aux(model, name='Turbine Flow',cache=False, jit=False)
1988.      def turbine_flow(t):
1989.          #IF THEN ELSE(Components collapsing gate=1, IF THEN ELSE(Head Cover=
    0, Head Cover Max Flow , IF THEN ELSE(Unit availability=1, Turbine instructions
    , 0)), 0)
1990.          if components_collapsing_gate(t)==1:
1991.              if head_cover(t)==0:
1992.                  return head_cover_max_flow(t)
1993.              if head_cover(t)==1 and unit_availability(t)==1:
1994.                  return turbine_instructions(t)
1995.              else: return 0
1996.          else:
1997.              return 0
1998.
1999.      @sdpy.aux(model, name="Powerhouse Flow Conveyance",cache=False, jit=Fals
    e)
2000.      def powerhouse_flow_conveyance(t):
2001.          return turbine_flow(t)
2002.
2003.      """
2004.      5. MODEL RUNNING (Base Case)
2005.
2006.      Stocks redefined each loop to ensure initial values are reset.
2007.      This may be changed later so stocks are also defined within their sector
    s above.
2008.
2009.      """
2010.      for yr in range(NYr):
2011.          year=yr
2012.          # Define time parameters to run model
2013.          initial_time = 0
2014.          final_time = 364
2015.          time_step = 1
2016.          model.run(initial_time, final_time, time_step)
2017.      #    print("Completed year :"+str(yr), flush=(yr%args.flush_period==0))

2018.          initial_reservoir_storage=SSCrev(B_RSEs[0,year])
2019.          if initial_reservoir_storage<=-
    304.1:  #making sure initial reservoir level isn't a failure
2020.              initial_reservoir_storage=364.27
2021.          model.reinitStock(initial_reservoir_storage, reservoir_storage)
2022.          model.reinitStock(np.zeros(3), gate_remaining_time_to_repair)
```

```
2023.           model.reinitStock(np.zeros(2), power_remaining_time_to_repair)
2024.           model.reinitStock(np.zeros(3), other_component_remaining_time_to_rep
    air)
2025.           model.reinitStock(0, sensor_remaining_time_to_repair)
2026.           model.reinitStock(1, time_remaining_to_access_site)
2027.           model.reinitStock(1, time_remaining_to_contact_staff)
2028.           model.reinitStock(0, manual_actuation_initiated)
2029.           plantStaffNotified=np.zeros(365)
2030.           RESEL_IG=[]
2031.
2032.
2033.       """
2034.       6. DAM SAFETY PRIORITIZED RUN
2035.
2036.       """
2037.
2038.       Output=[RSEs, TBFs, SPOGs, OT]
2039.       Otheroutput=[TTRS, TOTR, DEBRISREMOVAL, DAY, MON]
2040.       EOCs=RSEs-376.5
2041.       EOCs[EOCs<0]=0 #Filling in elevations over the core and truncating to ze
    ro if less than 376.5
2042.
2043.       @njit
2044.       def OTC(elev): #Overtopping curve
2045.           if elev<=378.41:
2046.               return 0
2047.           else:
2048.               return (-35.7505780379803*elev**3 + 40896.2749435669* elev**2 -
    15593240.0619064*elev + 1981715583.08889)
2049.
2050.
2051.       RESEL_IG=[]
2052.       @sdpy.aux(model, name="Intake Gate Closure",cache=False, jit=False)
2053.       def intake_gate_closure(t):
2054.           penstockrup=other_component_remaining_time_to_repair(t)[1]
2055.           hcfail=power_remaining_time_to_repair(t)[0]
2056.           igclosed=0
2057.           timestep=t
2058.           if hcfail>0 or penstockrup>0:
2059.               if timestep>OCdeltat[1,year]+1:
2060.                   igclosed=1 #closes immediately after 1 timestep
2061.           return igclosed
2062.
2063.       @sdpy.aux(model, name="Head Cover Max Flow",cache=False, jit=False)
2064.       def head_cover_max_flow(t): #reduces to 1/24th of actual release to acco
    unt for intake gate closure under rupture flow
2065.           flag1=head_cover(t)+1 #if head cover = 0 then flag=1
2066.           if other_component_remaining_time_to_repair(t)[1]>0: #penstock
2067.               flag1=1
2068.           resels=reservoir_level(t)
2069.           t1=fncTurbineMaxFlow(resels, flag1)
2070.           if intake_gate_closure(t)==1:
2071.               return 0
2072.           if intake_gate_closure(t)==0:
2073.               return (1/24.)*np.max([np.min([5*(t1), reservoir_storage(t)+rese
    rvoir_inflow(t)-gated_spill_release(t)-(-48.6)]), 0])
2074.
2075.       @sdpy.aux(model, name='Operations Planning',cache=False, jit=False)
2076.       @sdpy.subscript(controls)
2077.       def operations_planning(t):
2078.           timestep=t
```

```
2079.           dayref=dayrefs(Startdays[year], timestep)
2080.           Inf114=Inflow[timestep:14+timestep, year] #Changed to one day ahead
    so proper spills are released for Vensim version
2081.           InfForecast=Inf114
2082.           gaugerelay=gauge_relay(t)
2083.           if gaugerelay<-
    900 or gaugerelay>381.73: #If error is so high that it becomes obvious
2084.               storage=-1000
2085.           else:
2086.               storage=SSCrev(gaugerelay)
2087.           StaffOnSite=site_staff_mobilized(t)
2088.           actualstorage=reservoir_storage(t)
2089.           if (StaffOnSite>0) or (storage==-1000):
2090.               if storage==-1000:
2091.                   storage=SSCrev(RSEs[timestep-
    1, year]) #if unknown, takes previous days value
2092.               if StaffOnSite==1:
2093.                   storage=actualstorage  #if someone is on site, takes actual
    value
2094.           InitialStorage=np.float64(storage)#+lastinf-outfs
2095.           resElPens=np.zeros((3,3))#Penalties for res el
2096.           resElPens[0,:]=[0,273,304]
2097.           resElPens[1,:]=[426.99, 300.39, 300.39]
2098.           resElPens[2,:]=[99.58693574984267,99.58693574984267, 171.28] #123.60
    856547318923 from 87.055 to help reduce 0 spill events
2099.           SPOG1Av=gate_all(t)   #Availbility set based on "gate time to repair
    "
2100.           TurbAv1=power_all(t)
2101.           if SPOG1Av>0 or other_component_remaining_time_to_repair(t)[0]>0 or
    other_component_remaining_time_to_repair(t)[2]>0:
2102.               #if grid, plc or gate unavailable
2103.               resElPens[1,:]=[100, 100, 100] #reducing target nmax to 367.8 to
     keep reservoir low for large inflow events
2104.           Nextday=np.zeros(2)
2105.
2106.           Optimized=OpsPlan(InfForecast, InitialStorage, dayref,  SPOG1Av,  Tu
    rbAv1,  resElPens)
2107.           if SPOG1Av>0 or other_component_remaining_time_to_repair(t)[0]>0 or
    other_component_remaining_time_to_repair(t)[2]>0:
2108.               if TurbAv1<=0:
2109.                   Optimized[1]=fncTurbineMaxFlow(SSC(InitialStorage), 1)
2110.           Nextday=np.array(Optimized.copy())   #SPOG1, Turb1
2111.           Nextday.clip(min=0) #omit negatives.
2112.           return Nextday
2113.
2114.
2115.       #indices = np.random.choice(np.arange(NYr), replace=False, size=int(NYr
    * 0.5))
2116.       #OCOutages[0,indices] = 0
2117.       #OCdeltat[0,indices] = 0
2118.       #indices = np.random.choice(np.arange(NYr), replace=False, size=int(NYr
    * 0.5))
2119.       #SErrors[indices]=0
2120.       #SErrorDeltat[indices]=0
2121.       #indices = np.random.choice(np.arange(NYr), replace=False, size=int(NYr
    * 0.5))
2122.       #SOutages[indices]=0
2123.       #Sdeltat[indices]=0
2124.
2125.
2126.       #Save simulation 1 results with 1 in them.
```

```
2127.       RSEs1=RSEs
2128.       TBFs1=TBFs
2129.       SPOGs1=SPOGs
2130.       OT1=OT
2131.       OUTFs1=OUTFs
2132.       TTRS1=TTRS
2133.       CAPs1=CAPs
2134.       EOCs1=EOCs
2135.       UCRs1=UCRs
2136.       GCRs1=GCRs
2137.       GAVs1=GAVs
2138.       GPs1=GPs
2139.
2140.       #Redefining arrays for second run
2141.       RSEs=np.zeros((365,NYr))
2142.       GAVs=np.zeros((365,NYr))
2143.       TBFs=np.zeros((365,NYr))
2144.       SPOGs=np.zeros((365,NYr))
2145.       OT=np.zeros((365,NYr))
2146.       INFs=np.zeros((365,NYr))
2147.       OUTFs=np.zeros((365,NYr))
2148.       GPs=np.zeros((365, NYr))
2149.       TOTR=np.zeros((365,NYr))
2150.       DEBRISREMOVAL=np.zeros(NYr)
2151.       DAY=np.zeros((365,NYr))
2152.       MON=np.zeros((365,NYr))
2153.       AllMaxQ_t=np.zeros((365,2, NYr))
2154.       AllMaxQ=[861.1+728.9,32.5+32.5]
2155.       TTRS=np.zeros((365,8, NYr))
2156.       Retention=np.zeros((365,NYr))
2157.       yearnum=np.zeros(NYr)
2158.       for yr in range(NYr):
2159.           yearnum[yr]=str(1984+yr)
2160.       CAPs=np.zeros((365, NYr))
2161.       EOCs=np.zeros((365, NYr))
2162.       UCRs=np.zeros((365,NYr))
2163.       GCRs=np.zeros((365,NYr))
2164.
2165.       #Reducing sensor issues, plcrtu failures by 50%
2166.       OCOutages1=OCOutages.copy()
2167.       OCdeltat1=OCdeltat.copy()
2168.       SErrors1=SErrors.copy()
2169.       SErrorDeltat1=SErrorDeltat.copy()
2170.       SOutages1=SOutages.copy()
2171.       Sdeltat1=Sdeltat.copy()
2172.       GateOutagesAll1=GateOutagesAll.copy()
2173.
2174.       indices = np.random.choice(np.arange(NYr), replace=False, size=int(NYr *
      0.5))
2175.       OCOutages[0,indices] = 0
2176.       OCdeltat[0,indices] = 0
2177.       indices = np.random.choice(np.arange(NYr), replace=False, size=int(NYr *
      0.5))
2178.       SErrors[indices]=0
2179.       SErrorDeltat[indices]=0
2180.       indices = np.random.choice(np.arange(NYr), replace=False, size=int(NYr *
      0.5))
2181.       SOutages[indices]=0
2182.       Sdeltat[indices]=0
2183.       indices = np.random.choice(np.arange(NYr), replace=False, size=int(NYr *
      0.2))
```

```
2184.       GateOutagesAll[0,indices] = 0 #gate failing closed 15% improvement
2185.       indices = np.random.choice(np.arange(NYr), replace=False, size=int(NYr *
     0.2))
2186.       GateOutagesAll[2,indices] = 0 #gate failing in place 15% improvement
2187.
2188.
2189.       for yr in range(NYr):
2190.           tm=time.time()
2191.           year=yr
2192.           initial_reservoir_storage=SSCrev(B_RSEs[0,year])
2193.           if initial_reservoir_storage<=-
    304.1:  #making sure initial reservoir level isn't a failure
2194.               initial_reservoir_storage=364.27
2195.           model.reinitStock(initial_reservoir_storage, reservoir_storage)
2196.           model.reinitStock(np.zeros(3), gate_remaining_time_to_repair)
2197.           model.reinitStock(np.zeros(2), power_remaining_time_to_repair)
2198.           model.reinitStock(np.zeros(3), other_component_remaining_time_to_rep
    air)
2199.           model.reinitStock(0, sensor_remaining_time_to_repair)
2200.           model.reinitStock(1, time_remaining_to_access_site)
2201.           model.reinitStock(1, time_remaining_to_contact_staff)
2202.           model.reinitStock(0, manual_actuation_initiated)
2203.
2204.           # Define time parameters to run model
2205.           initial_time = 0
2206.           final_time = 364
2207.           time_step = 1
2208.
2209.
2210.           model.run(initial_time, final_time, time_step)
2211.       #    print("Completed DS year :"+str(yr), flush=(yr%args.flush_period==0
    ))
2212.       #    tm1=time.time()
2213.       #    timer.append(tm1-tm)
2214.           plantStaffNotified=np.zeros(365)
2215.           RESEL_IG=[]
2216.
2217.
2218.       """
2219.       7. POST-PROCESSING AND SAVING RESULTS
2220.       Percentiles are saved to reduce output file sizes as much as possible
2221.
2222.       """
2223.
2224.       S_RL=pd.read_csv(name1)
2225.       S_RL=S_RL.set_index("NewInd")
2226.       S_CL=pd.read_csv(name2)
2227.       S_CL=S_CL.set_index("NewInd") #setting index to the formatted OS IDs
2228.
2229.       scenar = all_scenarios[seednum]
2230.       ScenarioRL=S_RL.filter(items=scenar[0:7], axis=0)
2231.       ScenarioCL=S_CL.filter(items=scenar[7:13], axis=0)
2232.       AbnormalRL=ScenarioRL[ScenarioRL['CausalFactorName']!="None"]
2233.       AbnormalCL=ScenarioCL[ScenarioCL['CausalFactorName']!="None"]
2234.       AbnormalCL=AbnormalCL[AbnormalCL['CausalFactorName']!="Normal"]
2235.
2236.       ScenarioIDs=[]
2237.
2238.       for i in range(len(AbnormalRL)):
2239.           ScenarioIDs.append(AbnormalRL.index[i])
2240.       for i in range(len(AbnormalCL)):
```

```
2241.           ScenarioIDs.append(AbnormalCL.index[i])
2242.
2243.       AllAdScenarios=ScenarioIDs.copy()
2244.
2245.       AllTimes=np.zeros((11, NYr))
2246.       AllTimes[0:3,:]=Gdeltat
2247.       AllTimes[3:5,:]=Tdeltat
2248.       AllTimes[5,:]=Sdeltat1
2249.       AllTimes[6:9,:]=OCdeltat1
2250.       if "1359_2" in ScenarioIDs: #adding debris
2251.           AllTimes[9,:]=np.ones(NYr)
2252.       AllTimes[10,:]=IFErrorDeltat
2253.       AllTimes=AllTimes.transpose()
2254.       ColNames=["1362", "1361", "1360", "836", "838","30", "18","42","44", "13
    59", "45"]
2255.       AllTimes=pd.DataFrame(AllTimes, columns=ColNames)
2256.
2257.       PersonnelScenarios=[]
2258.       if "48_1" in ScenarioIDs:
2259.           PersonnelScenarios.append("48")
2260.           ScenarioIDs.remove("48_1")
2261.       if "48_2" in ScenarioIDs:
2262.           PersonnelScenarios.append("48")
2263.           ScenarioIDs.remove("48_2")
2264.       if "29_1" in ScenarioIDs:
2265.           PersonnelScenarios.append("29")
2266.           ScenarioIDs.remove("29_1")
2267.       if "29_2" in ScenarioIDs:
2268.           PersonnelScenarios.append("29")
2269.           ScenarioIDs.remove("29_2")
2270.       if "29_3" in ScenarioIDs:
2271.           PersonnelScenarios.append("29")
2272.           ScenarioIDs.remove("29_3")
2273.
2274.
2275.
2276.       ScenarioIDs_simp=[]
2277.       for i in range(len(ScenarioIDs)):
2278.           head, sep, tail = ScenarioIDs[i].partition('_')
2279.           ScenarioIDs_simp.append(head)
2280.
2281.       AdTimes=AllTimes[ScenarioIDs_simp]
2282.
2283.       TrueScenarios1=[] #this will contain a list of true scenario results
2284.       for yr in range(NYr):
2285.           imptimes=AdTimes.iloc[yr]
2286.           imptimesarr=np.array(imptimes)
2287.           RSEdiff=1 #this means there is a difference between the normal and c
    ase reservoir levels
2288.           scenar=[]
2289.           for t in range(361):
2290.               if t in imptimesarr:
2291.                   if len(scenar)==0:
2292.                       scenstart=t
2293.                   imps=imptimes[imptimes == t].index
2294.                   for i in range(len(imps)):
2295.                       scenar.append(imps[i])       #will keep adding to this s
    cenario as events happen, if RSEs don't change
2296.               #NOW check for scenario end, which happens when the next 3 days
    RSE is within 0.05 m of normal
2297.               if len(scenar)!=0:
```

```
                if (-0.05<(RSEs1[t+1,yr]-B_RSEs[t+1, yr])<0.05) and (-0.05<(RSEs1[t+2,yr]-B_RSEs[t+2, yr])<0.05) and (-0.05<(RSEs1[t+3,yr]-B_RSEs[t+3, yr])<0.05) or (RSEs1[t,yr]<=353.75) or (t==360):
                    RSEdiff=0
                    scenend=t
                    if len(PersonnelScenarios)>0:
    #                    if imps[i]=='44' or imps[i]=='18': #plc/rtu or grid failures necessitate site access for gate op
                        scenar+=PersonnelScenarios #add site and staff delays to ensure they are counted towards scenario

                    if scenstart!=scenend:
                    #post process sub-scenario
                        Failure=0
                        RSEs1subset=RSEs1[scenstart:t+1, yr]
                        SPOGs1subset=SPOGs1[scenstart:t+1, yr]
                        TBFs1subset=TBFs1[scenstart:t+1, yr]
                        OTs1subset=OT1[scenstart:t+1, yr]

                        CAPs1subset=CAPs1[scenstart:t+1, yr]
                        UCRs1subset=UCRs1[scenstart:t+1, yr]
                        GCRs1subset=GCRs1[scenstart:t+1, yr]
                        if np.sum(UCRs1subset)==0:
                            UCRs1subset=0 #avoid saving useless info
                        if min(CAPs1subset)==1655:
                            CAPs1subset=1655 #avoid saving useless info
                        if min(GCRs1subset)==2:
                            GCRs1subset=2 #avoid saving useless info
                        INFsubset=INFs[scenstart:t+1, yr]
                        Avg5dInfThreshold=0
                        Max5dInfThreshold=0
                        if min(RSEs1subset)<=353.75:
                            Failure=1
                            minind=np.argmin(RSEs1subset)
                            if minind>5:
                                Avg5dInfThreshold=np.mean(INFsubset[minind-5:minind])
                                Max5dInfThreshold=max(INFsubset[minind-5:minind])
                            else:
                                Avg5dInfThreshold=np.mean(INFsubset[0:minind])
                                Max5dInfThreshold=max(INFsubset[0:minind])
                        maxRSE=max(RSEs1subset)
                        #replacing elements in scenar with complete OS identifier
                        scenar1= {pref:ele for pref in scenar for ele in AllAdScenarios if pref in ele}
                        scenar1 = list(scenar1.values())
                        AllOS=['18_3', '29_4', '30_4', '42_2', '44_3', '45_2', '48_3', '836_2', '838_2', '1359_1', '1360_3', '1361_1', '1362_1']
                        for i in range(len(scenar1)): #convert all normal to the scenario represented in scenar1
                            head, sep, tail = scenar1[i].partition('_')
                            indices = [i for i, s in enumerate(AllOS) if head in s]
                            AllOS[indices[0]]=scenar1[i] #complete list of OS's
                        #Convert list of OS's to seed number
                        subseednum=all_scenarios.get_scenario_index(AllOS)
```

```
                            sOut1=[subseednum, scenar1, Starts[yr], (scenstart, scenend), Failure, (Avg5dInfThreshold, Max5dInfThreshold), maxRSE, RSEs1subset, CAPs1subset, UCRs1subset, GCRs1subset, yr, SPOGs1subset, TBFs1subset, OTs1subset]
                            #reset scenar
                            scenar=[]
                            TrueScenarios1.append(sOut1)
                            if RSEs1[t,yr]<=353.75: #eliminate unnecessary further processing
                                break
                        else:
                            scenar=[] #skips scenarios that didn't cause any difference in reservoir levels

        # Dam safety improved

        AllTimes=np.zeros((11, NYr))
        AllTimes[0:3,:]=Gdeltat
        AllTimes[3:5,:]=Tdeltat
        AllTimes[5,:]=Sdeltat
        AllTimes[6:9,:]=OCdeltat
        if "1359_2" in ScenarioIDs: #adding debris
            AllTimes[9,:]=np.ones(NYr)
        AllTimes[10,:]=IFErrorDeltat
        AllTimes=AllTimes.transpose()
        ColNames=["1362", "1361", "1360", "836", "838","30", "18","42","44", "1359", "45"]
        AllTimes=pd.DataFrame(AllTimes, columns=ColNames)


        ScenarioIDs_simp=[]
        for i in range(len(ScenarioIDs)):
            head, sep, tail = ScenarioIDs[i].partition('_')
            ScenarioIDs_simp.append(head)

        AdTimes=AllTimes[ScenarioIDs_simp]

        TrueScenarios=[] #this will contain a list of true scenario results
        for yr in range(NYr):
            imptimes=AdTimes.iloc[yr]
            imptimesarr=np.array(imptimes)
            RSEdiff=1 #this means there is a difference between the normal and case reservoir levels
            scenar=[]
            for t in range(361):
                if t in imptimesarr:
                    if len(scenar)==0:
                        scenstart=t
                    imps=imptimes[imptimes == t].index
                    for i in range(len(imps)):
                        scenar.append(imps[i])       #will keep adding to this scenario as events happen, if RSEs don't change
                #NOW check for scenario end, which happens when the next 3 days RSE is within 0.05 m of normal
                if len(scenar)!=0:
                    if (-0.05<(RSEs[t+1,yr]-B_RSEs[t+1, yr])<0.05) and (-0.05<(RSEs[t+2,yr]-B_RSEs[t+2, yr])<0.05) and (-0.05<(RSEs[t+3,yr]-B_RSEs[t+3, yr])<0.05) or (RSEs[t,yr]<=353.75) or (t==360):
                        RSEdiff=0
                        scenend=t
                        if len(PersonnelScenarios)>0:
```

```
2395.      #                    if imps[i]=='44' or imps[i]=='18': #plc/rtu or grid
    failures necessitate site access for gate op
2396.                          scenar+=PersonnelScenarios #add site and staff delay
    s to ensure they are counted towards scenario
2397.                      if scenstart!=scenend:
2398.                          RSEdiff=0
2399.                          scenend=t
2400.                          #post process sub-scenario
2401.                          Failure=0
2402.                          RSEs1subset=RSEs[scenstart:t+1, yr]
2403.                          CAPs1subset=CAPs[scenstart:t+1, yr]
2404.                          UCRs1subset=UCRs[scenstart:t+1, yr]
2405.                          GCRs1subset=GCRs[scenstart:t+1, yr]
2406.                          if np.sum(UCRs1subset)==0:
2407.                              UCRs1subset=0 #avoid saving useless info
2408.                          if min(CAPs1subset)==1655:
2409.                              CAPs1subset=1655 #avoid saving useless info
2410.                          if min(GCRs1subset)==2:
2411.                              GCRs1subset=2 #avoid saving useless info
2412.                          INFsubset=INFs[scenstart:t+1, yr]
2413.                          Avg5dInfThreshold=0
2414.                          Max5dInfThreshold=0
2415.                          if min(RSEs1subset)<=353.75:
2416.                              Failure=1
2417.                              minind=np.argmin(RSEs1subset)
2418.                              if minind>5:
2419.                                  Avg5dInfThreshold=np.mean(INFsubset[minind-
    5:minind])
2420.                                  Max5dInfThreshold=max(INFsubset[minind-
    5:minind])
2421.                              else:
2422.                                  Avg5dInfThreshold=np.mean(INFsubset[0:minind
    ])
2423.                                  Max5dInfThreshold=max(INFsubset[0:minind])
2424.                          maxRSE=max(RSEs1subset)
2425.                          #replacing elements in scenar with complete OS ident
    ifier
2426.                          scenar1= {pref:ele for pref in scenar for ele in All
    AdScenarios if pref in ele}
2427.                          scenar1 = list(scenar1.values())
2428.                          AllOS=['18_3', '29_4', '30_4', '42_2', '44_3', '45_2
    ', '48_3', '836_2', '838_2', '1359_1', '1360_3', '1361_1', '1362_1']
2429.                          for i in range(len(scenar1)): #convert all normal to
     the scenario represented in scenar1
2430.                              head, sep, tail = scenar1[i].partition('_')
2431.                              indices = [i for i, s in enumerate(AllOS) if hea
    d in s]
2432.                              AllOS[indices[0]]=scenar1[i] #complete list of O
    S's
2433.                          #Convert list of OS's to seed number
2434.                          subseednum=all_scenarios.get_scenario_index(AllOS)
2435.                          sOut=[subseednum, scenar1, Starts[yr], (scenstart, s
    cenend), Failure, (Avg5dInfThreshold, Max5dInfThreshold), maxRSE, RSEs1subset, C
    APs1subset, UCRs1subset, GCRs1subset, yr]
2436.                          #reset scenar
2437.                          scenar=[]
2438.                          TrueScenarios.append(sOut)
2439.                          if RSEs[t,yr]<=353.75: #eliminate unnecessary furthe
    r processing
2440.                              break
2441.                      else:
```

```
                            scenar=[] #skips scenarios that didn't cause any dif
    ference in reservoir levels


        #Error messages
        #Check min and max RSEs
        err=[]
        mnrse=np.min(RSEs)
        if mnrse<320:
            err.append("Minimum RSE below el. 320m")
        mxspog=np.max(SPOGs)
        if mxspog>1590:
            err.append("Gate flow exceeds 1590 maximumum")


        #reorganizing outputs
        numscen=len(TrueScenarios1)
        seednums1=np.zeros(numscen)
        seedstarts1=np.zeros((numscen,2))
        scendates1=np.zeros((numscen,2))
        failures1=np.zeros(numscen)
        infthresh1=np.zeros((numscen,2))
        years1=np.zeros(numscen)
        maxrse1=np.zeros(numscen)
        for i in range(numscen):
            seednums1[i]=TrueScenarios1[i][0]
            seedstarts1[i,:]=TrueScenarios1[i][2]
            scendates1[i,:]=TrueScenarios1[i][3]
            failures1[i]=TrueScenarios1[i][4]
            infthresh1[i,:]=TrueScenarios1[i][5]
            maxrse1[i]=TrueScenarios1[i][6]
            years1[i]=TrueScenarios1[i][11]

        seedfailures1=0
        for i in range(numscen):
            if seednums1[i]==seednum:
                seedfailures1+=failures1[i]
        seedsim1=np.count_nonzero(seednums1==seednum)


        TSIterations1=np.where(seednums1==seednum)[0]
        if len(TSIterations1)>0:
            scenariolengthmax=int(np.max(scendates1[:,1]-scendates1[:,0]))+1
            RSEs1all=np.zeros((len(TSIterations1), scenariolengthmax))
            RSEs1all[RSEs1all==0]='nan'
            SPOGs1all=np.zeros((len(TSIterations1), scenariolengthmax))
            SPOGs1all[SPOGs1all==0]='nan'
            TBFs1all=np.zeros((len(TSIterations1), scenariolengthmax))
            TBFs1all[TBFs1all==0]='nan'
            OTs1all=np.zeros((len(TSIterations1), scenariolengthmax))
            OTs1all[OTs1all==0]='nan'

            CAPs1all=np.zeros((len(TSIterations1), scenariolengthmax))
            CAPs1all[CAPs1all==0]='nan'
            UCRs1all=np.zeros((len(TSIterations1), scenariolengthmax))
            UCRs1all[UCRs1all==0]='nan'
            GCRs1all=np.zeros((len(TSIterations1), scenariolengthmax))
            GCRs1all[GCRs1all==0]='nan'
            for j in range(len(TSIterations1)):
                i=TSIterations1[j]
```

```
                scenariolength=int(scendates1[i,1]-scendates1[i,0])
                RSEs1all[j,0:len(TrueScenarios1[i][7])]=TrueScenarios1[i][7]
                if RSEs1all[j,len(TrueScenarios1[i][7])-1]<=353.75:
                    RSEs1all[j, len(TrueScenarios1[i][7]):scenariolengthmax]=353
    .75 #count breach all the way to the end for plotting
                try:
                    CAPs1all[j,0:len(TrueScenarios1[i][8])]=TrueScenarios1[i][8]

                except:
                    CAPs1all[j,0]=TrueScenarios1[i][8]
                try:
                    UCRs1all[j,0:len(TrueScenarios1[i][9])]=TrueScenarios1[i][9]

                except:
                    UCRs1all[j,0]=TrueScenarios1[i][9]
                try:
                    GCRs1all[j,0:len(TrueScenarios1[i][10])]=TrueScenarios1[i][1
    0]
                except:
                    GCRs1all[j,0]=TrueScenarios1[i][10]
                try:
                    SPOGs1all[j,0:len(TrueScenarios1[i][12])]=TrueScenarios1[i][
    12]
                except:
                    SPOGs1all[j,0]=TrueScenarios1[i][12]
                try:
                    TBFs1all[j,0:len(TrueScenarios1[i][13])]=TrueScenarios1[i][1
    3]
                except:
                    TBFs1all[j,0]=TrueScenarios1[i][13]
                try:
                    OTs1all[j,0:len(TrueScenarios1[i][14])]=TrueScenarios1[i][14
    ]
                except:
                    OTs1all[j,0]=TrueScenarios1[i][14]

        else:
            RSEs1all=np.array(["nan","nan"])
            SPOGs1all=np.array(["nan","nan"])
            TBFs1all=np.array(["nan","nan"])
            OTs1all=np.array(["nan","nan"])
            CAPs1all=np.array(["nan","nan"])
            UCRs1all=np.array(["nan","nan"])
            GCRs1all=np.array(["nan","nan"])


        #reorganizing outputs
        numscen=len(TrueScenarios)
        seednums=np.zeros(numscen)
        seedstarts=np.zeros((numscen,2))
        scendates=np.zeros((numscen,2))
        failures=np.zeros(numscen)
        infthresh=np.zeros((numscen,2))
        maxrse=np.zeros(numscen)
        years=np.zeros((numscen,2))
        for i in range(numscen):
            seednums[i]=TrueScenarios[i][0]
            seedstarts[i,:]=TrueScenarios[i][2]
            scendates[i,:]=TrueScenarios[i][3]
            failures[i]=TrueScenarios[i][4]
            infthresh[i,:]=TrueScenarios[i][5]
```

```
            maxrse[i]=TrueScenarios[i][6]
            years[i]=TrueScenarios[i][11]

        seedfailures=0
        for i in range(numscen):
            if seednums[i]==seednum:
                seedfailures+=failures[i]
        seedsim=np.count_nonzero(seednums==seednum)

        TSIterations=np.where(seednums==seednum)[0]
        if len(TSIterations)>0:
            scenariolengthmax=int(np.max(scendates[:,1]-scendates[:,0]))+1
            RSEsall=np.zeros((len(TSIterations), scenariolengthmax))
            RSEsall[RSEsall==0]='nan'
            CAPsall=np.zeros((len(TSIterations), scenariolengthmax))
            CAPsall[CAPsall==0]='nan'
            UCRsall=np.zeros((len(TSIterations), scenariolengthmax))
            UCRsall[UCRsall==0]='nan'
            GCRsall=np.zeros((len(TSIterations), scenariolengthmax))
            GCRsall[GCRsall==0]='nan'
            for j in range(len(TSIterations)-1):
                i=TSIterations[j]
                scenariolength=int(scendates[i,1]-scendates[i,0])
                RSEsall[j,0:len(TrueScenarios[i][7])]=TrueScenarios[i][7]
                if RSEsall[j,len(TrueScenarios[i][7])-1]<=353.75:
                    RSEsall[j, len(TrueScenarios1[i][7]):scenariolengthmax]=353.
    75 #count breach all the way to the end for plotting
                try:
                    CAPsall[j,0:len(TrueScenarios[i][8])]=TrueScenarios[i][8]
                except:
                    CAPsall[j,0]=TrueScenarios[i][8]
                try:
                    UCRsall[j,0:len(TrueScenarios[i][9])]=TrueScenarios[i][9]
                except:
                    UCRsall[j,0]=TrueScenarios[i][9]
                try:
                    GCRsall[j,0:len(TrueScenarios[i][10])]=TrueScenarios[i][10]

                except:
                    GCRsall[j,0]=TrueScenarios[i][10]

        else:
            RSEsall=np.array(["nan","nan"])
            SPOGsall=np.array(["nan","nan"])
            TBFsall=np.array(["nan","nan"])
            OTsall=np.array(["nan","nan"])
            CAPsall=np.array(["nan","nan"])
            UCRsall=np.array(["nan","nan"])
            GCRsall=np.array(["nan","nan"])


        #Write txt output file
        if len(err)==0:
            np.savez_compressed(str("Outputs-"+str(seednum)+".npz"),
                     seednums1=seednums1,
                     seedstards1=seedstarts1,
                     scendates1=scendates1,
                     failures1=failures1,
                     infthresh1=infthresh1,
                     maxrse1=maxrse1,
```

```
                    seedfailures1=seedfailures1,
                    seedsim1=seedsim1,
                    RSEs1all=RSEs1all.transpose(),
                    CAPS1all=CAPs1all.transpose(),
                    UCRs1all=UCRs1all.transpose(),
                    GCRs1all=GCRs1all.transpose(),
                    seednums=seednums,
                    seedstards=seedstarts,
                    scendates=scendates,
                    failures=failures,
                    infthresh=infthresh,
                    maxrse=maxrse,
                    seedfailures=seedfailures,
                    seedsim=seedsim,
                    RSEsall=RSEsall.transpose(),
                    CAPSall=CAPsall.transpose(),
                    UCRsall=UCRsall.transpose(),
                    GCRsall=GCRsall.transpose()
                )


        if len(err)>0:
            np.savez(str("Outputs-"+str(seednum)+"-e.npz"),
                    seednums1=seednums1,
                    seedstards1=seedstarts1,
                    scendates1=scendates1,
                    failures1=failures1,
                    infthresh1=infthresh1,
                    maxrse1=maxrse1,
                    seedfailures1=seedfailures1,
                    seedsim1=seedsim1,
                    RSEs1all=RSEs1all.transpose(),
                    CAPS1all=CAPs1all.transpose(),
                    UCRs1all=UCRs1all.transpose(),
                    GCRs1all=GCRs1all.transpose(),
                    seednums=seednums,
                    seedstards=seedstarts,
                    scendates=scendates,
                    failures=failures,
                    infthresh=infthresh,
                    maxrse=maxrse,
                    seedfailures=seedfailures,
                    seedsim=seedsim,
                    RSEsall=RSEsall.transpose(),
                    CAPSall=CAPsall.transpose(),
                    UCRsall=UCRsall.transpose(),
                    GCRsall=GCRsall.transpose()
                )

        t0_2=time.time()

        #print("elapsed time: " +str(t0_2-t0))
        #
        #
        #print("number of data points, base case:" +str(np.count_nonzero(seednum
    s1==seednum)))
        #
        #print("number of data points, DSI case:" +str(np.count_nonzero(seednums
    ==seednum)))
```

Appendix F: High Performance Computing

There are a total of 552,960 simulations, each simulated for 2000 iterations for two runs: the base case and the dam safety improved case. Each iteration lasts for one year, so there are a total of 2.2 Billion simulation-years. This is obviously a very large simulation exercise that requires HPC resources to be executed efficiently. Compute Canada offers several HPC clusters, and this research utilized Graham, Cedar and Niagara to complete the simulations. Each cluster has thousands of nodes and each node may have several cores. Because the scenarios are completely independent of one another, serial farming is the best implementation for efficient simulation for this project. Serial farming means that processes can run completely independently on multiple cores at a time, and their order of execution is not important. In order to set up the serial farming environment, a simulation controller is required.

The controller is a bash-scripted program that performs several functions and was developed with assistance from a programming consultant due to its complex nature. It is used to set up workspaces on the various clusters, and to send scenarios to the clusters in preparation for simulation. Once the simulations are on the cluster and ready for processing, the controller is used to initiate “jobs” which process the simulations on the cluster. When submitting a job, the user can specify the number of jobs, the number of cores to be used for each individual job as well as the time limit after which the job terminates. The controller also manages the list of scenarios which have been completed, submitted, failed or are still running and ensures there are no duplicating simulations for a single scenario. The status of the jobs in terms of the number of scenarios running, completed, and waiting in the queue can be queried by the user. Once jobs on Graham or Cedar clusters are finished, the controller automatically resubmits the jobs to continue processing the list of scenarios. The user must monitor the job status periodically, and ensure more scenarios are available on the list for continued processing. For the Niagara cluster, jobs are not able to re-submit themselves, so the user must manually submit either smaller numbers of large jobs or larger numbers of small jobs and monitor them. Finally, the controller is used to download the scenario output files to the local machine, which in this case is a virtual private server. Given the 10TB storage capacity, the *.npz compressed

array files contain only the key outputs – dynamic reservoir level response, criticality parameters and performance measures. These *.npz array files are stored in a directory that contains sub-folders with 1000 files each.

During the initial test run of the controller, some issues with the simulation model for specific scenarios were identified and repaired. The initial run of the model was completed over a three-week period. Compute Canada has a specific scheduling algorithm which allocates resources to users based on their priority as well as the amount of processing previously carried out. The jobs wait to start in a queueing system, and once a user's allocation is used up the priority of their jobs is reduced. This queuing system makes it difficult to estimate exactly what the throughput and simulation time will be. Resource allocations significantly improve throughput, and this was realized on the second complete simulation of the scenarios.

Curriculum Vitae

Name: Leanna M. King

Post-secondary Education and Degrees:

University of Western Ontario
London, Ontario, Canada
2007-2011 B.E.Sc.

The University of Western Ontario
London, Ontario, Canada
2011-2012 M.E.Sc.

Honours and Awards:

Alexander Graham Bell Canada Graduate Scholarship (NSERC CGS-D)
2016-2019

Ontario Graduate Scholarship
2011-2012, 2015

UWO Entrance Scholarship
2007

Related Work Experience

Engineer-in-Training
Dam Safety, BC Hydro
2019-Present

Teaching Assistant
The University of Western Ontario
2011-2012, 2015-2017

Engineer-in-Training
Hydrotechnical Division, BC Hydro
2013-2015

Publications:

King LM, Simonovic SP (2020) A Deterministic Monte Carlo simulation framework for dam safety flow-control assessment. Water 12(2): 505. doi: 10.3390/w12020505

King LM, Schardong A, Simonovic SP (2019) A combinatorial procedure to determine the full range of potential operating scenarios for a dam system. Water Resources Management 33(4):1451-1466. doi: 10.1007/s11269-018-2182-3

King LM, Simonovic SP, Hartford DND (2017) Using system dynamics simulation for assessment of hydropower system safety. Water Resources Research 53(8): 7148-7174. doi: 10.1002/2017WR020834

King LM, Keech S, Simonovic SP (2016) An Investigation of the Factors and Components Involved in Dam Safety Flow Control Incidents. Journal of Dam Engineering 27:1–19

King LM, McLeod AI, Simonovic SP. (2015) Improved Weather Generator Algorithm for Multisite Simulation of Precipitation and Temperature. Journal of the American Water Resources Association 51(5).

King LM, McLeod AI, Simonovic SP. (2014) Simulation of historical temperatures using a multi-site, multivariate block resampling algorithm with perturbation. Hydrological Processes 28(3).

King LM, Irwin S. Sarwar R, McLeod AI, Simonovic SP. (2012) The effects of climate change on extreme precipitation events in the upper Thames River Basin: A comparison of downscaling approaches. Canadian Water Resources Journal. 37(3).